TRACE32: the most complete tool for embedded linux debugging

- Maurizio Menegotto
  Lauterbach Italy

A Linux system is composed of several software components very different from each other. Free or cheap debuggers are generally used for one of these components, but not for others, and require the user to wade through different debugging techniques not homogeneous.

The aim of the presentation is to illustrate how a professional system Lauterbach TRACE32™ enable debugging of each linux component, from uboot to the kernel, modules and dynamic libraries, from processes to threads: a total view of the system, with a single debugger, in the same debugging session.

Agenda

Maurizio Menegotto
Lauterbach Italy

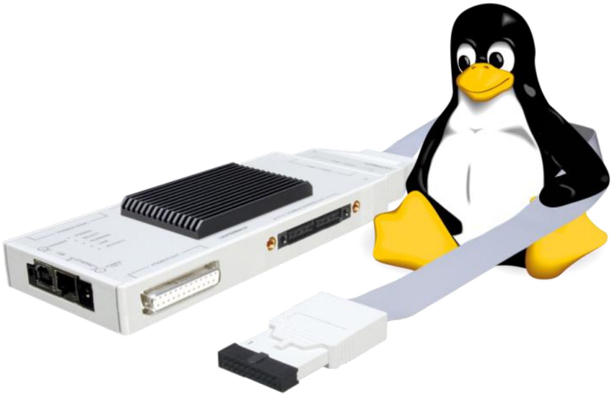
Seminar and live demo

- Linux debugging: problems & solution
- Debugging all linux components
- Stop-mode & run-mode debugging
- Trace, performance, profiling
- TRACE32 PowerTools
- Q&A

Agenda

Seminar and live demo

- **Linux debugging: problems & solution**
- Debugging all linux components
- Stop-mode & run-mode debugging
- Trace, performance, profiling
- TRACE32 PowerTools
- Q&A

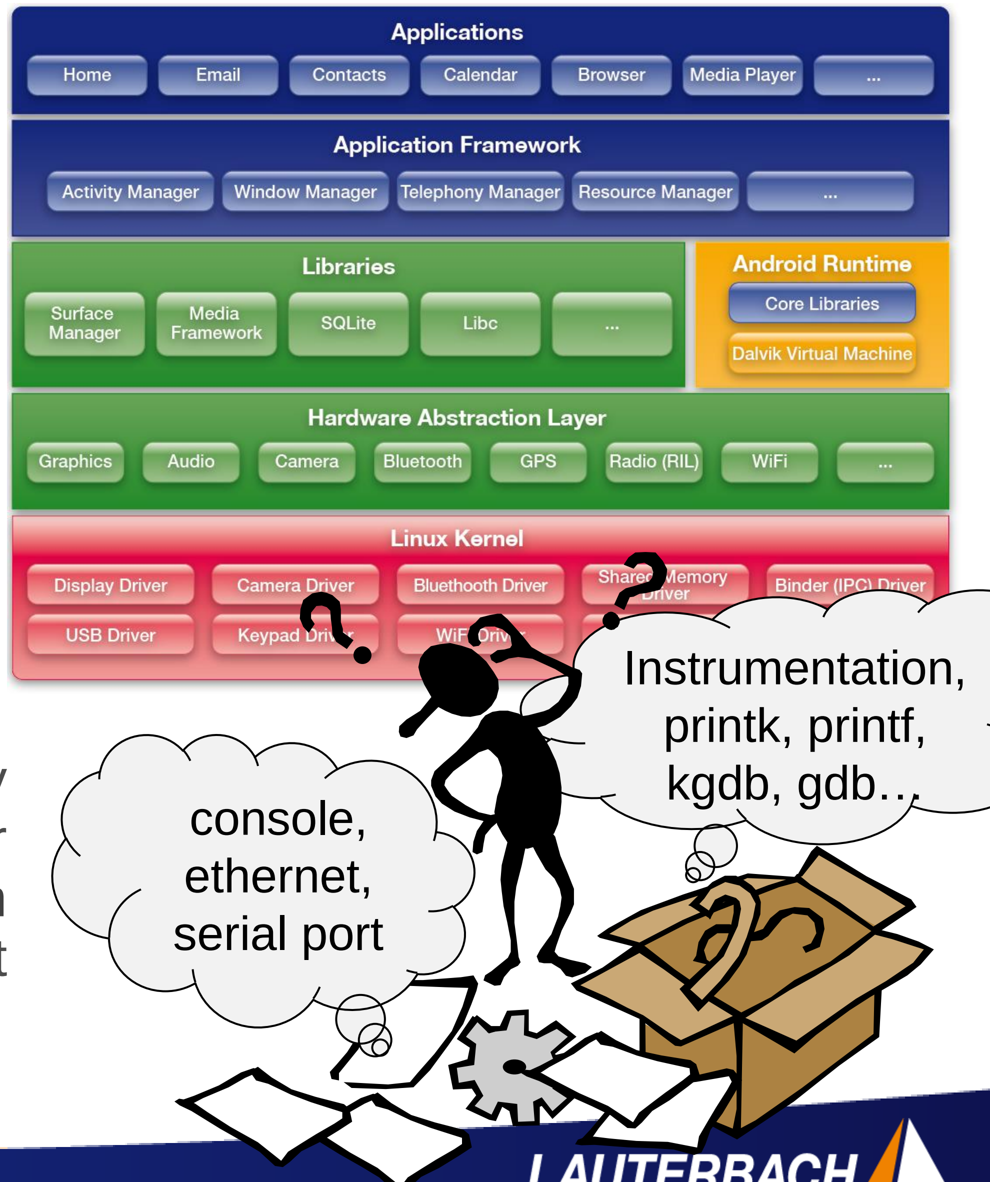


Linux debugging, the problems

In modern embedded systems, more and more frequently developers use operating systems, Linux is one of the most open source kernel used.

An embedded system based on Linux poses several problems from the point of view of debugging, as it consists of many different elements, and has advanced features that complicates the live of the debugger, such as on-demand paging, and dynamic MMU management.

Linux system components



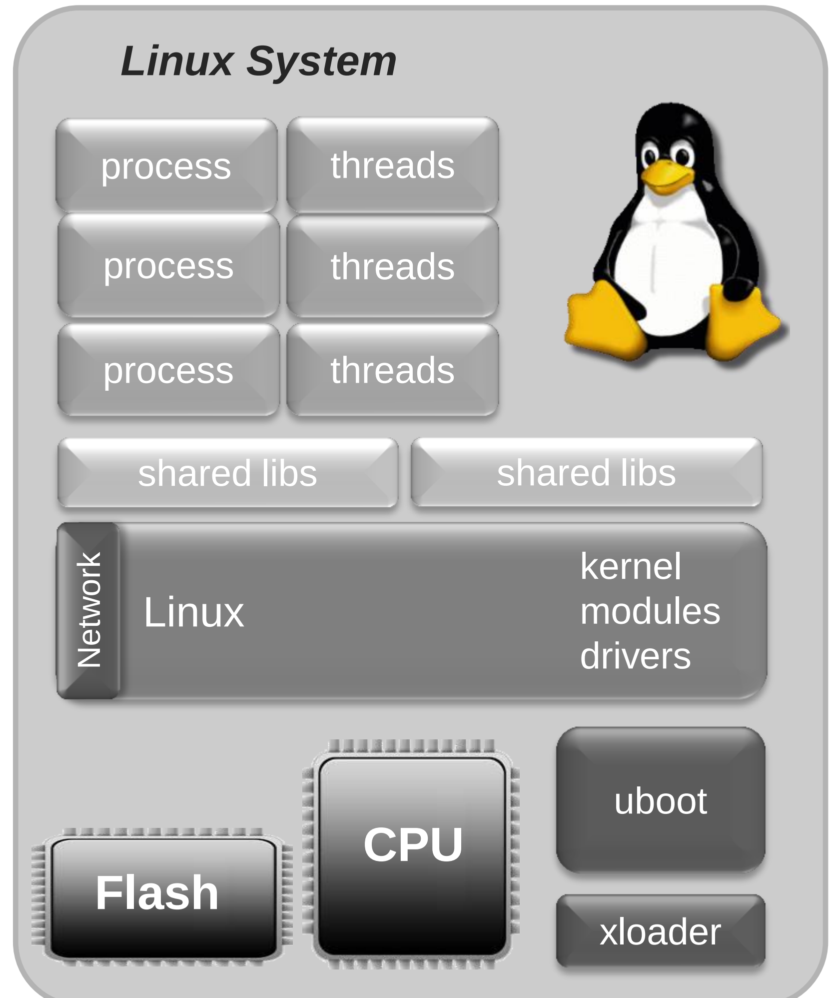
The debugger free or cheap are generally used for one of these components, but not for others, and require the user to wade through different debugging techniques and frequent recompilation.

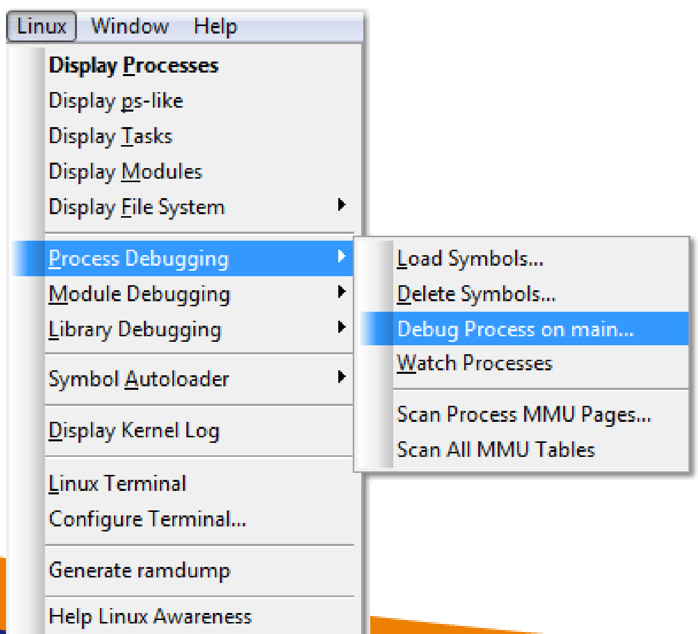
LAUTERBACH DEVELOPMENT TOOLS

Linux debugging, a unique solution

A professional system Lauterbach TRACE32 enable debugging of each linux component, from uboot to the kernel, modules and dynamic libraries, from processes to threads.

TRACE32 PowerView gives an immediate and complete view of the entire system with a single debugger, in the same debugging session, both in stop-mode and run-mode.





Linux debugging, what you need:

To debug a Linux system with TRACE32 you need:

Your computer

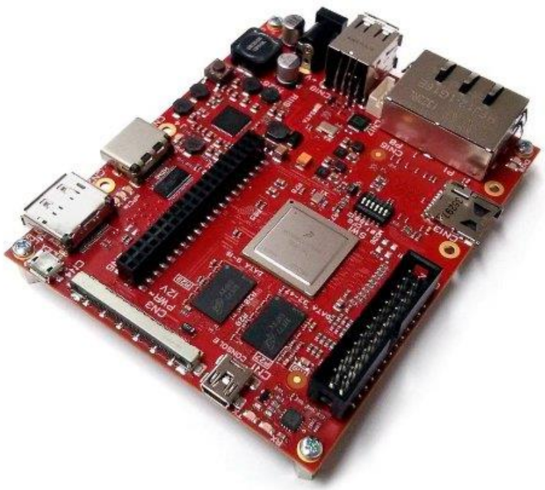
- PC Linux or Windows
- Your linux application
- TRACE32 PowerView SW

Your TRACE32

A PowerDebug HW + JTAG debug cable for your chip.

A Cortex™-A9 in this example

Your Target



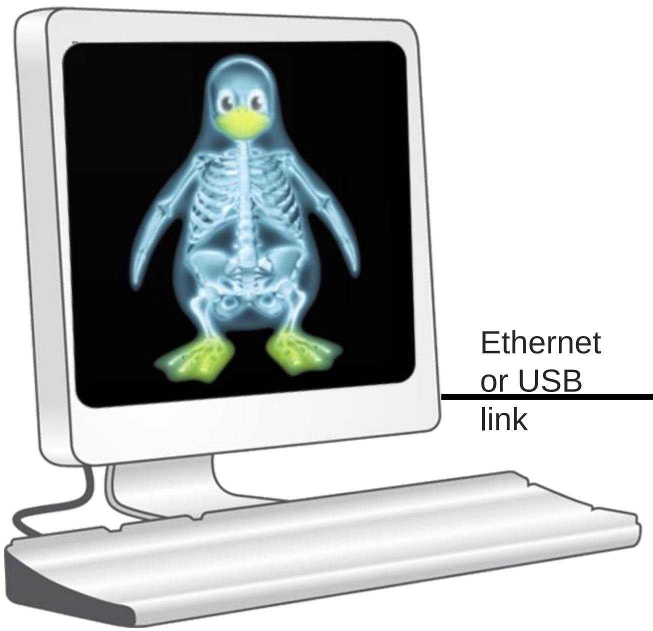
Target with a JTAG port

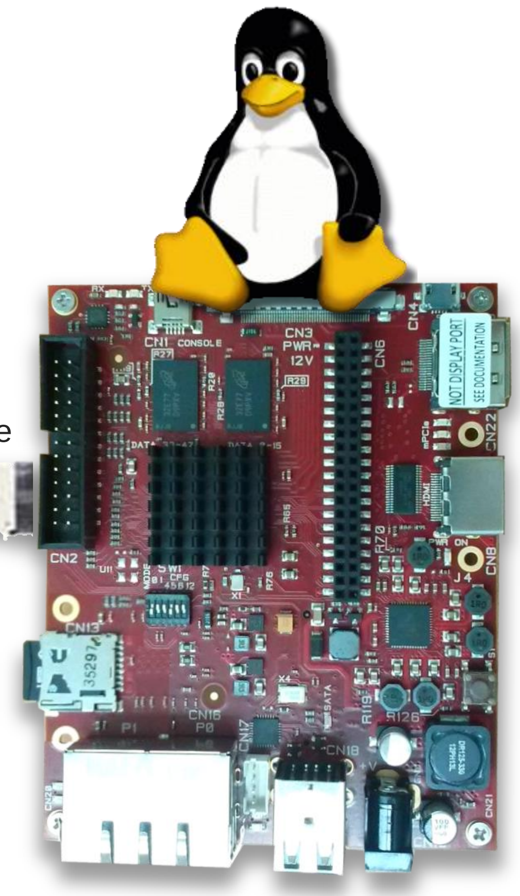
An Architech Tibidabo Board based on Freescale iMX6 Quad

➔ *Note: application and kernel must be compiled with debug symbols!*

Linux debugging, connection

The only physical connection to the target, required for debugging, is the JTAG port. TRACE32 has full target control since power-on reset.





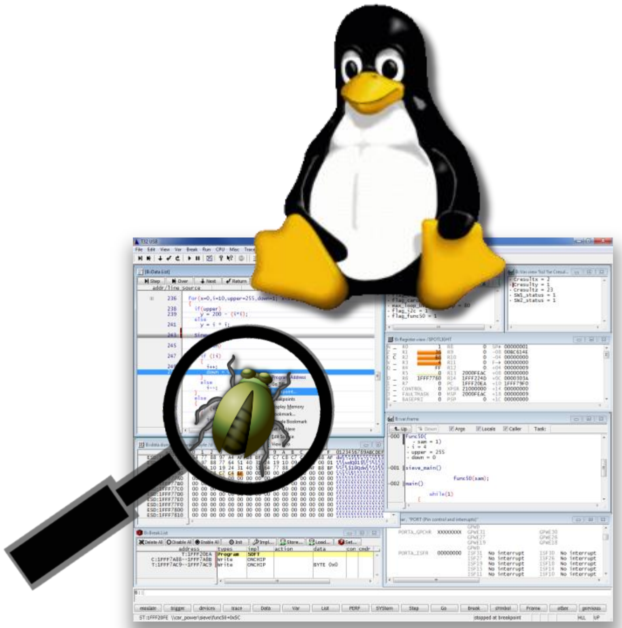
TRACE32 can debug a Linux system already stored in flash, or just load a new kernel into execution memory and start it.

LAUTERBACH DEVELOPMENT TOOLS

Agenda

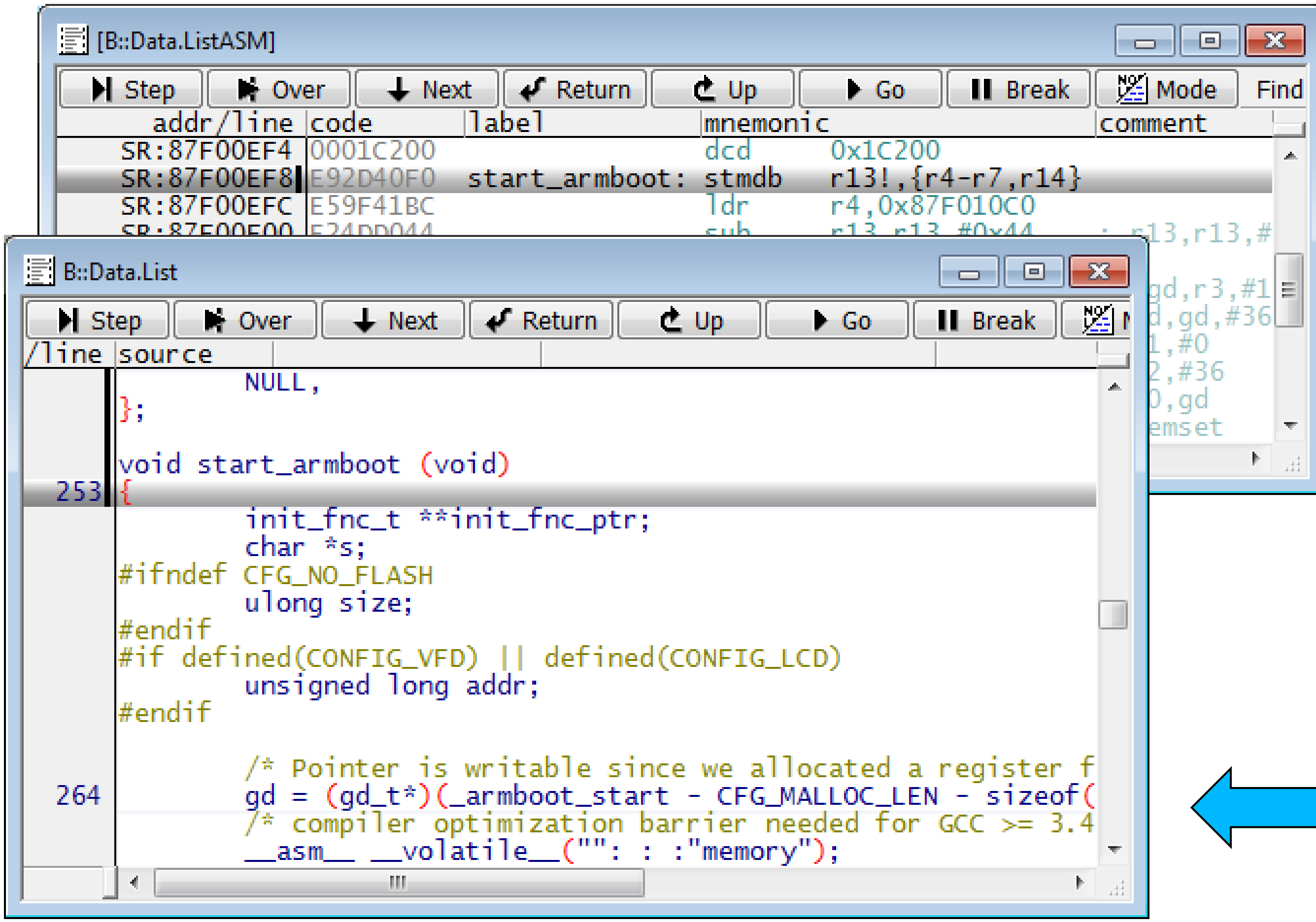
Seminar and live demo

- Linux debugging: problems & solution
- **Debugging all linux components**
- Stop-mode & run-mode debugging
- Trace, performance, profiling
- TRACE32 PowerTools
- Q&A

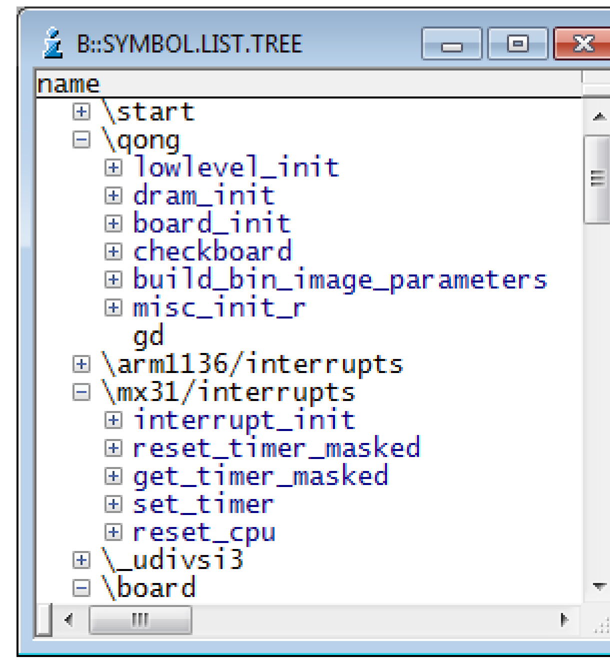


Debugging linux components: booting

Enabling the debugging session TRACE32 can take control of the cpu since the boot and show the program in memory, usually stopped at entry point



You can load symbols to debug the bootloader (Xloader, Uboot, ...)



And proceed to source-level debugging of everything is running before Linux

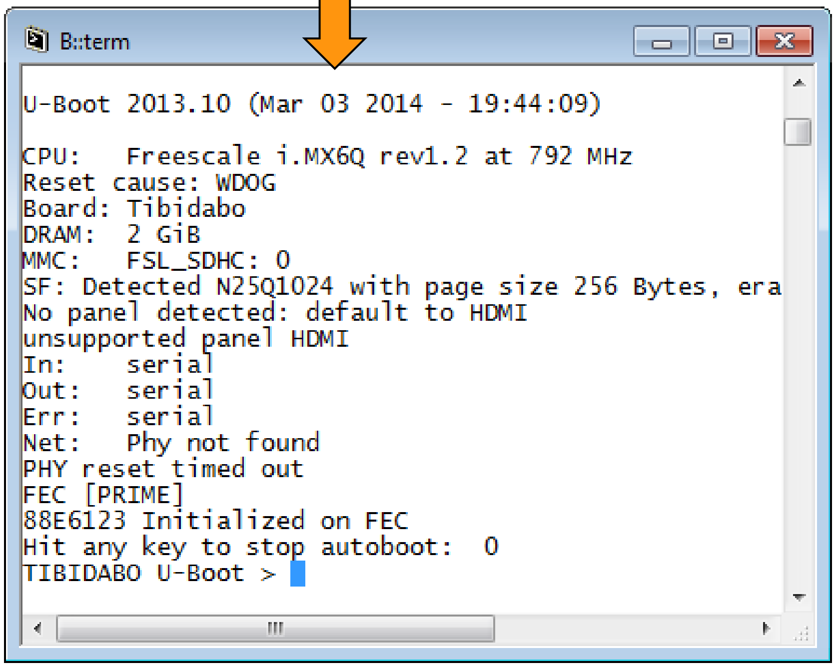
LAUTERBACH DEVELOPMENT TOOLS

Debugging linux components: kernel start

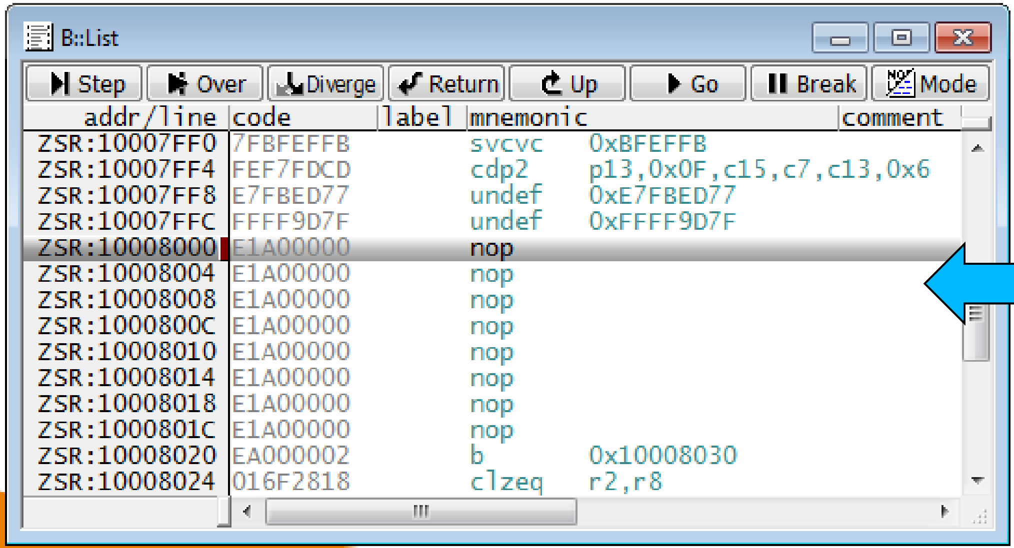
Terminal emulator integrated in TRACE32

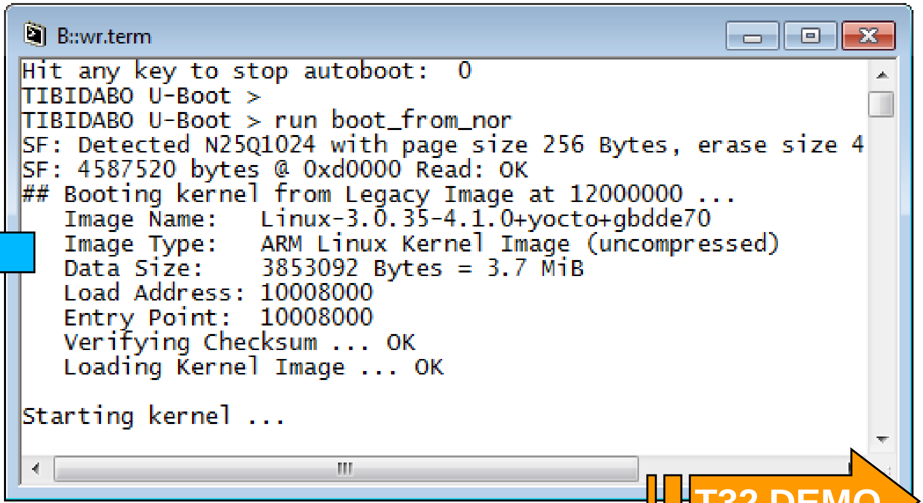
Typically a bootloader initializes the hardware and configure it to run the operating system.

The Linux kernel image (uImage) is loaded into RAM by the bootloader (uboot) or even by the debugger itself.



By booting the kernel, you can continue debugging from the entry point of Linux:



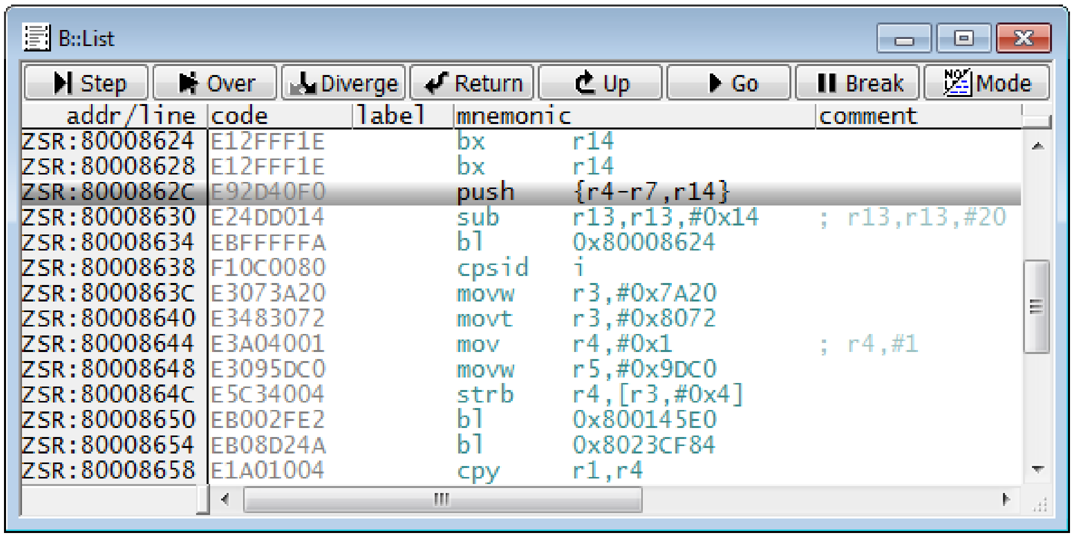


T32 DEMO

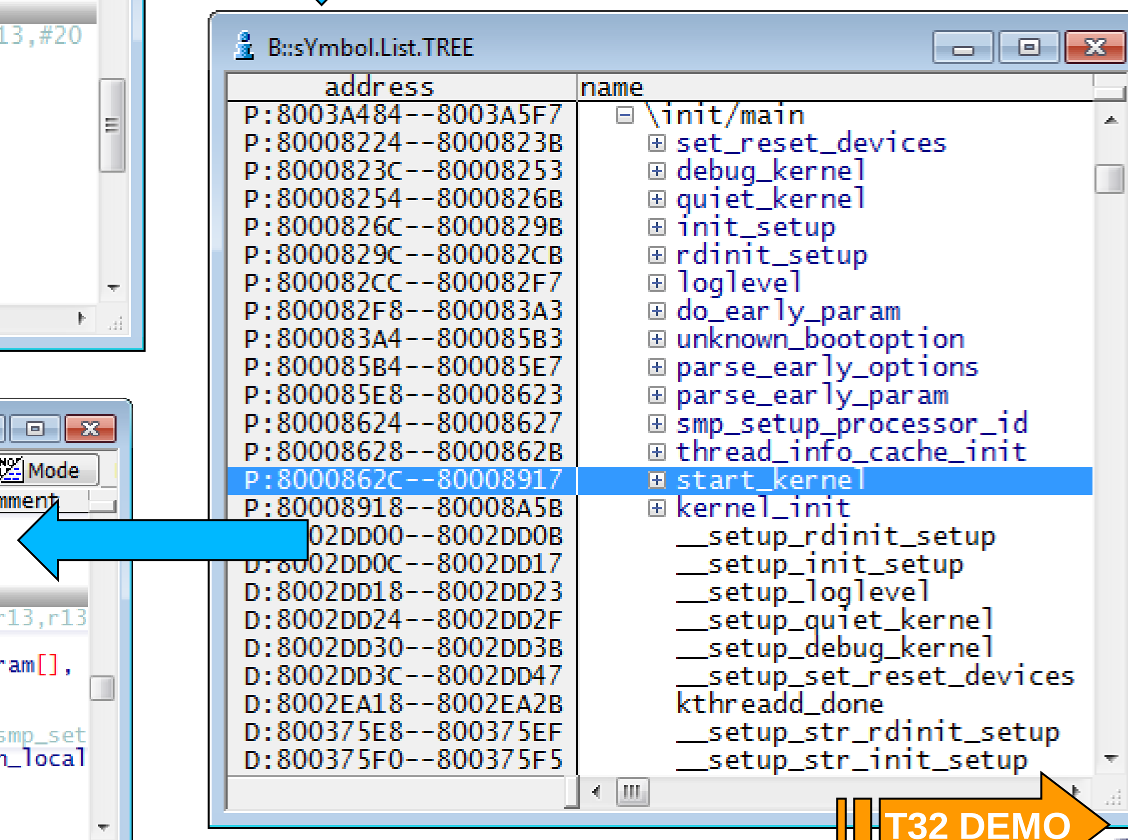
LAUTERBACH DEVELOPMENT TOOLS

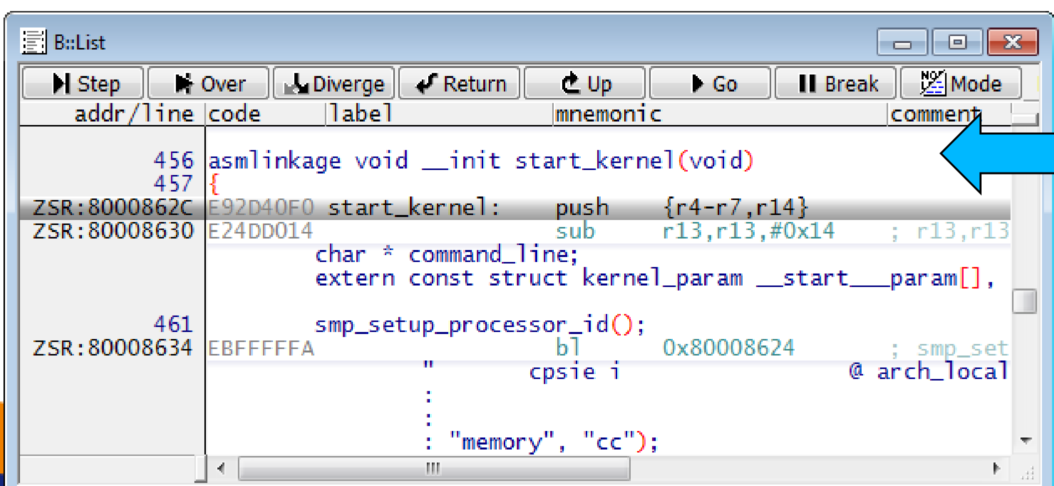
Debugging linux components: kernel symbols

If you boot Linux and stop the run with a BREAK, you can see the program running at address 0x8xxxxxxx



You can load kernel symbols (elf/dwarf file vmlinux) and debug it from start_kernel()



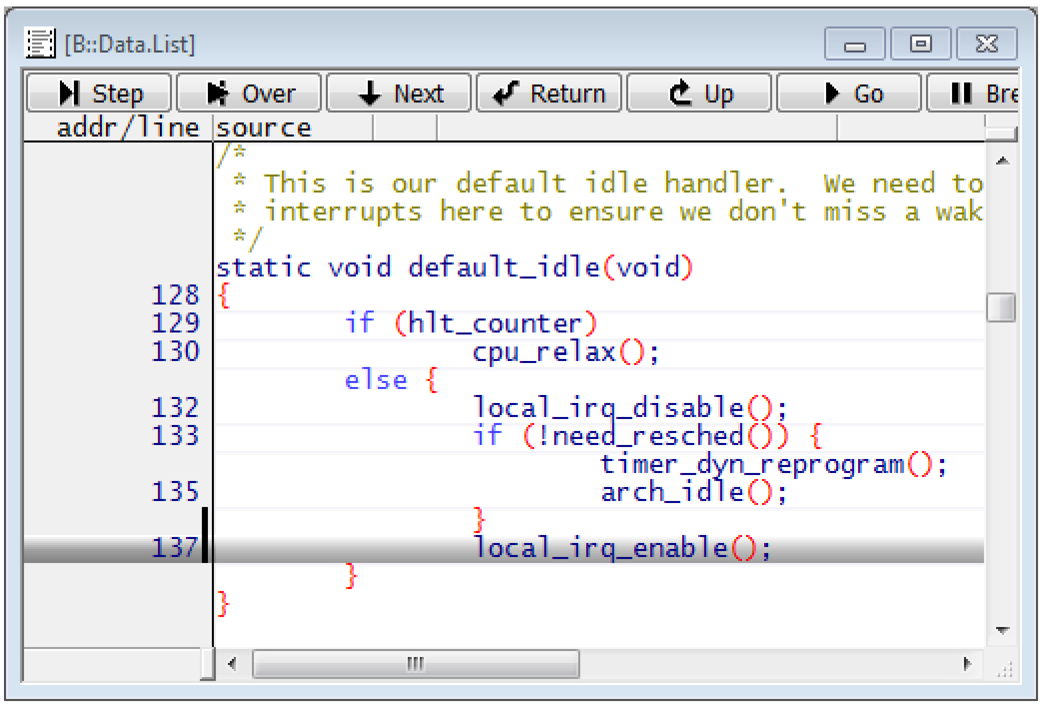


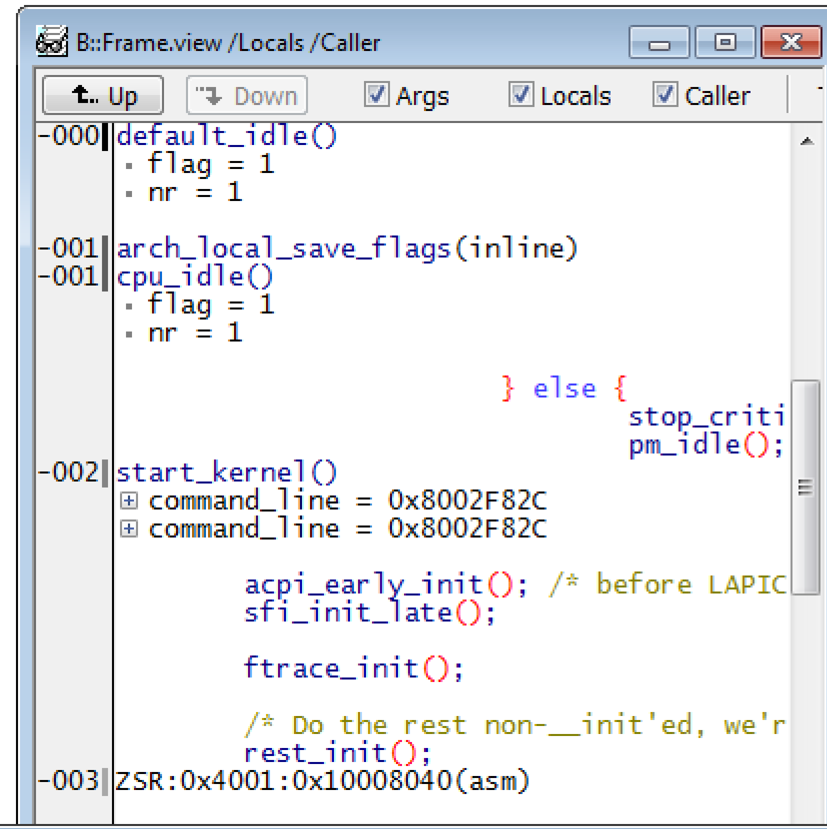
T32 DEMO

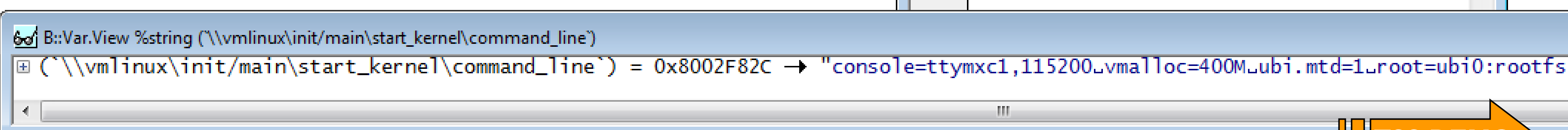
LAUTERBACH DEVELOPMENT TOOLS

Debugging linux components: kernel debugging

With the symbols (vmlinux) you can do source-level debugging of kernel: you can set breakpoints, run in step, see functions, registers, static variables and local variables in the stack-frame ...





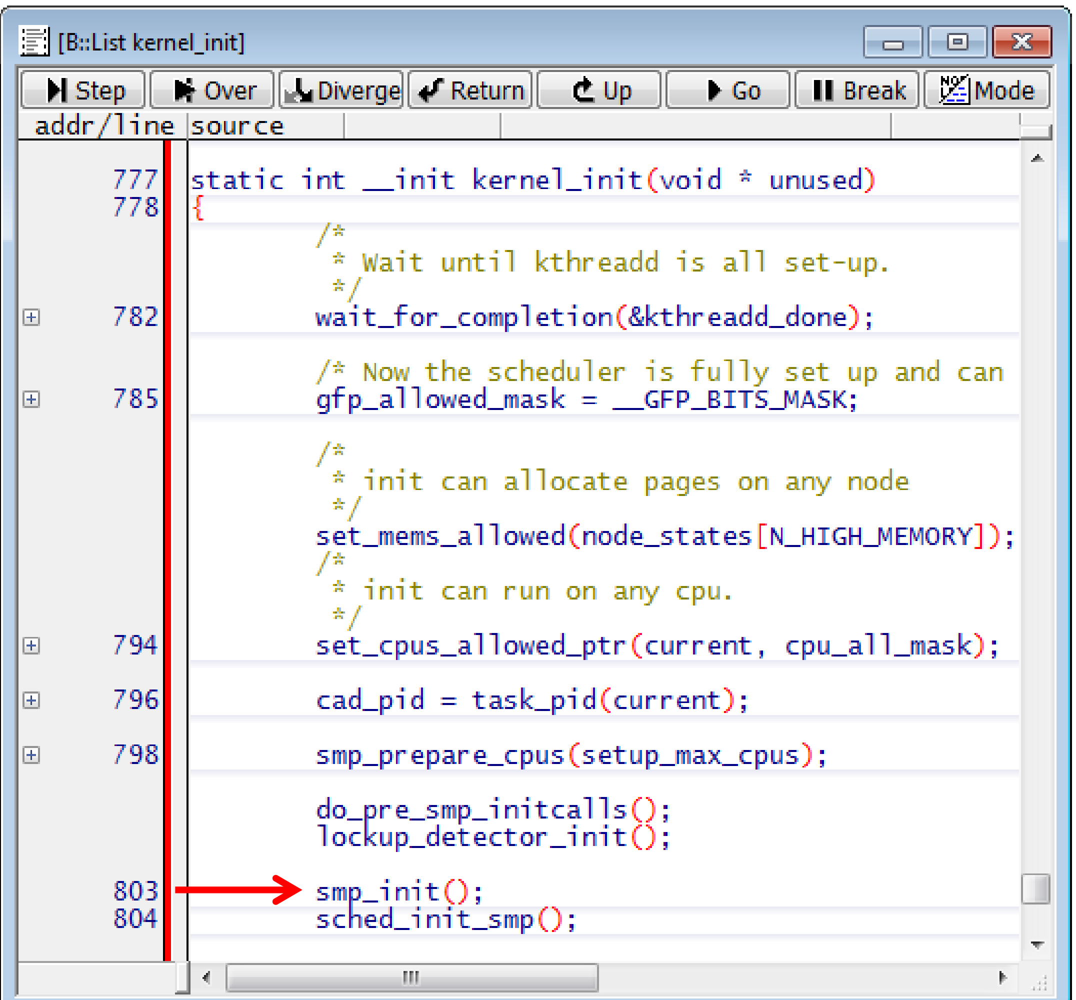


T32 DEMO

LAUTERBACH DEVELOPMENT TOOLS

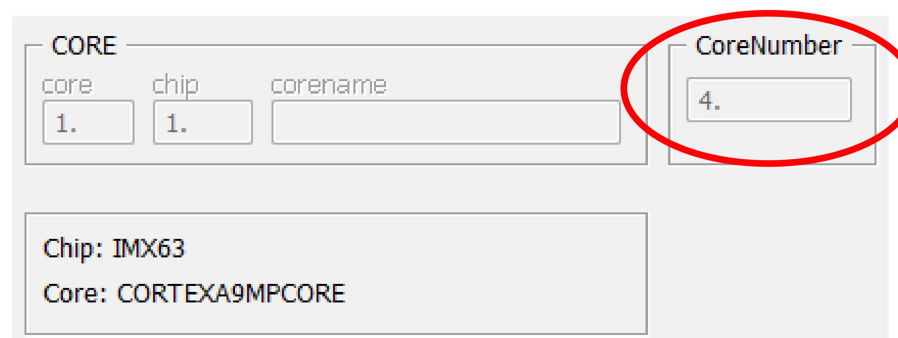
Debugging linux components: multicore debugging

The kernel is initializing with only core 0 active. In kernel_init() function is called smp_init() which activates the secondary cores 1, 2, 3.



The system becomes now multicore, so you must configure TRACE32 to handle 4 cores simultaneously in SMP mode (Symmetric Multi Processing):

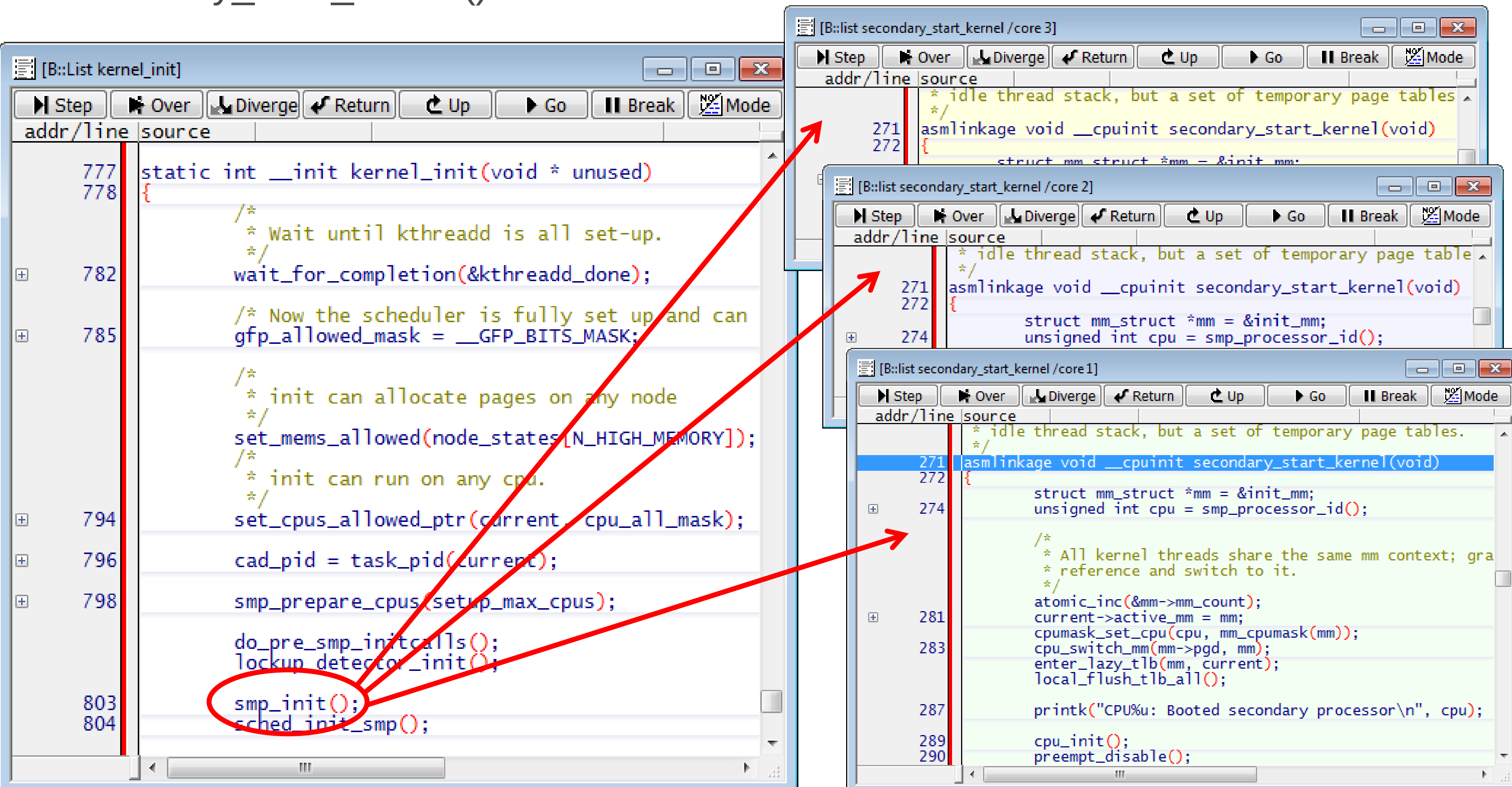
```
CORE.ASSIGN 1 2 3 4
```



LAUTERBACH DEVELOPMENT TOOLS

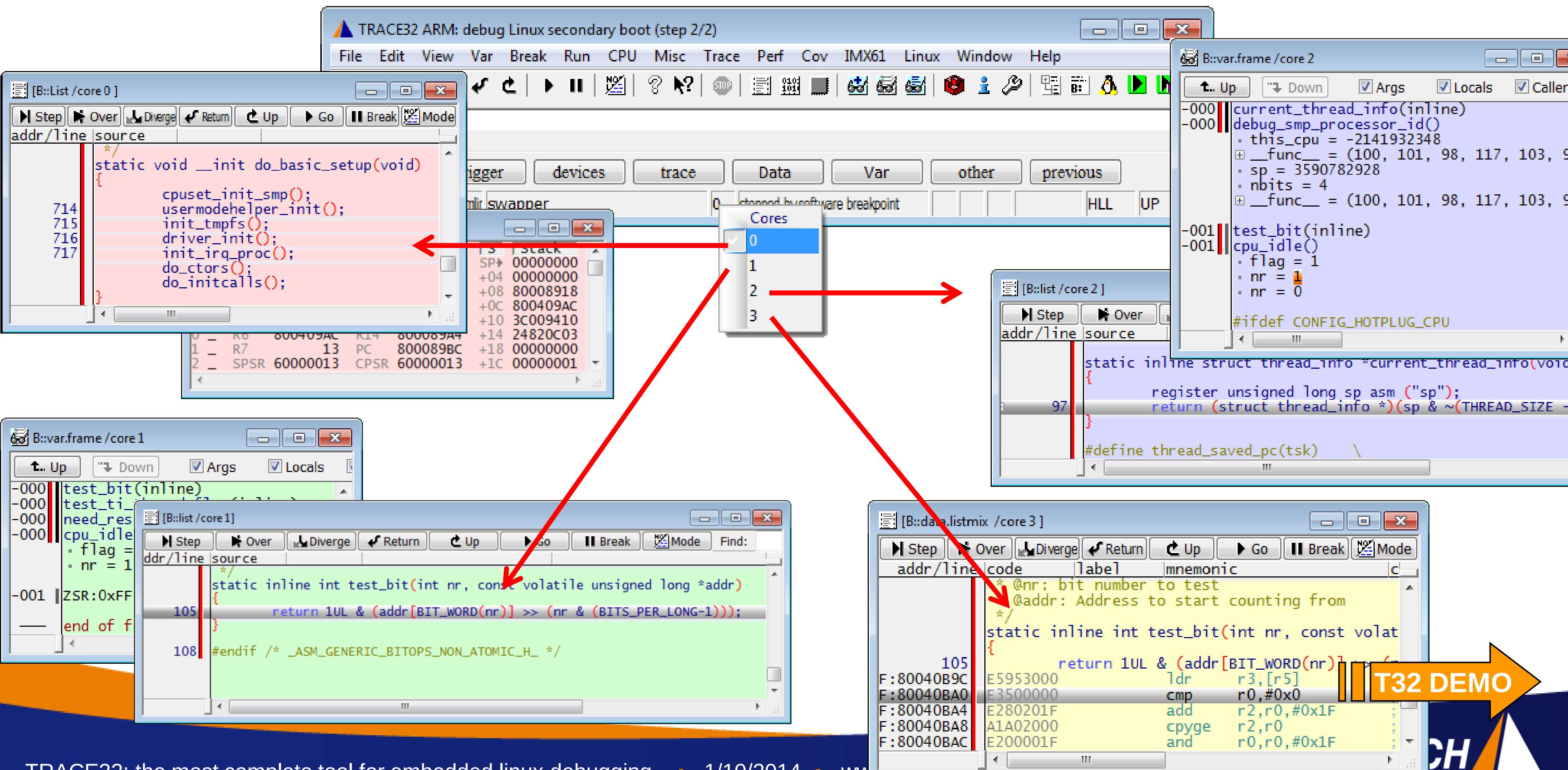
Debugging linux components: multicore debugging

After the execution of smp_init() and configuration of TRACE32 in multicore mode you can open new views on other cores, which will be running starting from secondary_start_kernel() function.



Debugging linux components: multicore debugging

In multicore mode SMP, all cores are handled simultaneously for commands as GO/BREAK/STEP and Breakpoints. For each window, you can have a specific view/color by core, or you can select the default core view in the Cores menu.



Debugging linux components: linux aware debugging

Kernel debugging can be done with a JTAG debugger also not specific to Linux. The entire kernel block can be considered as a single program (very big).

HOWEVER this is not enough to debug an entire Linux system

- ❑ How can you make the debugging of "dynamic objects" as the processes and their threads, libraries and kernel modules?
- ➢ You must consider the memory management (MMU) of the CPU and the kernel
- ➢ The debugger must be aware of the running operating system. Must give a view of the resources of Linux and have specific commands for their debug.

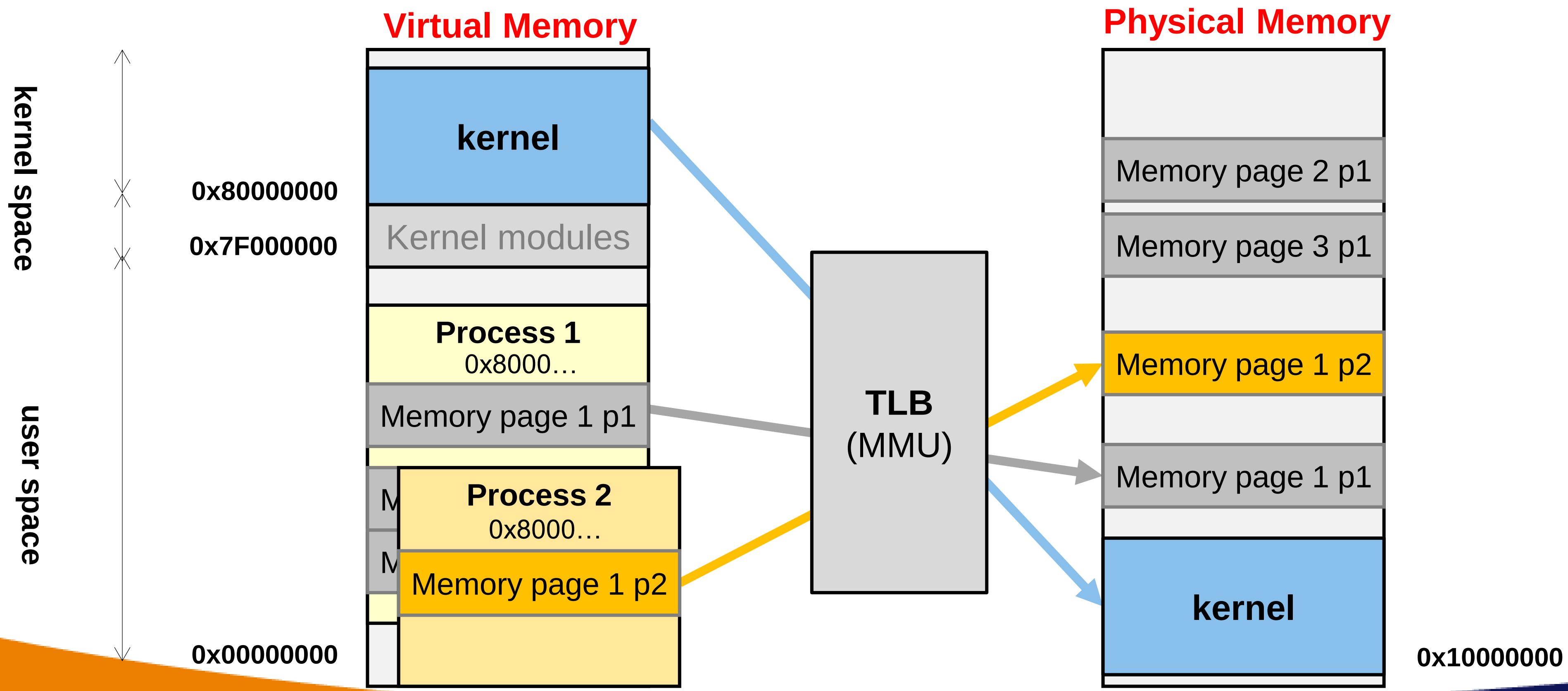
All of this is managed by the extension

TRACE32 Linux Awareness

(linux.t32)

Debugging linux components: Memory Management (1)

- The different components of the system are physically loaded in memory at absolute addresses, but execute at virtual address (logical)
- Kernel has a fixed Virtual-Physical translation
- Processes have dynamics Virtual-Physical translations



Debugging linux components: Memory Management (2)

- During debugging, the user uses logical addresses to access programs, data, symbols loaded into virtual memory *
- The core and the debugger can only access the active memory pages (TLB)
- TRACE32 can access the entire memory even at absolute addresses
- If a virtual memory area is not accessible, then TRACE32 computes the logical-physical translation, and make an access to physical address.

In this way TRACE32 allows the user to access at any time and debug in any area of memory using simple virtual addresses (logical)

* virtual addresses (logical) correspond to the program symbols

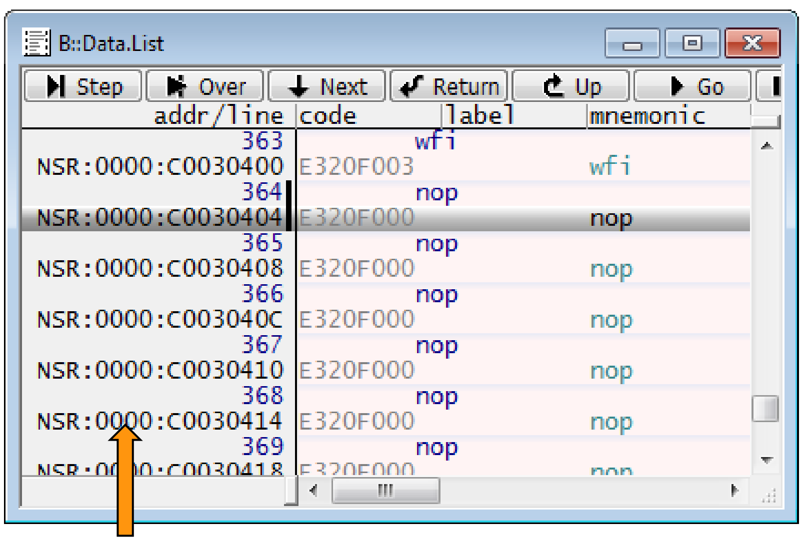
LAUTERBACH DEVELOPMENT TOOLS

Debugging linux components: Address Extension

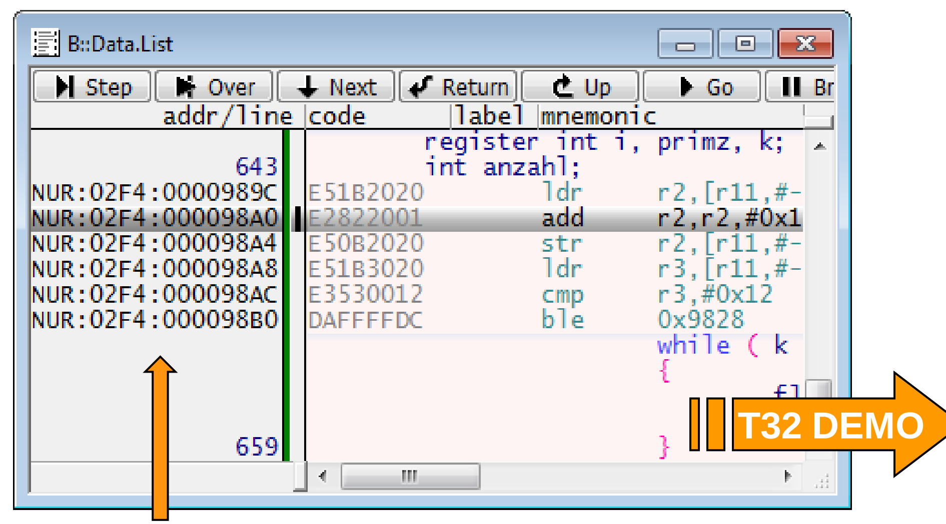
How to distinguish between kernel and process and between different processes?

- In Linux, the **space-id** of a process is the **PID** of his main thread
- The kernel and all its threads have by convention space-id = zero

- TRACE32 uses the identifier space-id to distinguish between the different processes by extending the address space.
- The Address Extension is enabled by the command
  SYStem.Option MMUSPACES ON



Space-id = **0x0000** : kernel thread

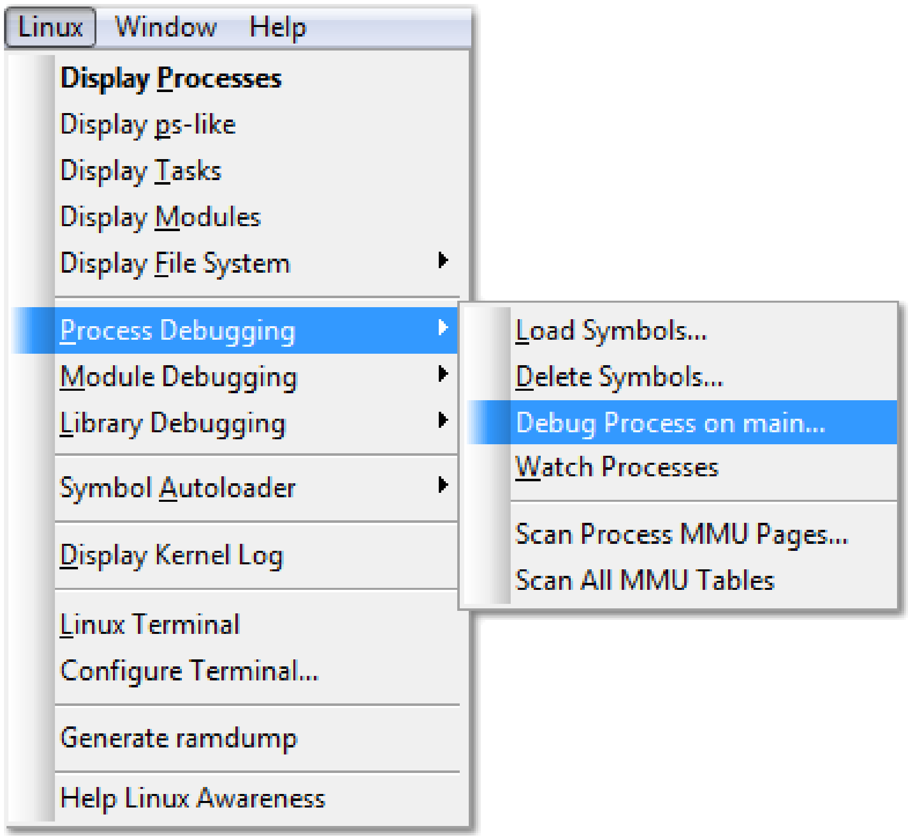


Space-id = **0x02F4** : user process PID 0x02F4

LAUTERBACH DEVELOPMENT TOOLS

Debugging linux components: linux menu

Thanks to Memory Management and Address Extension TRACE32 allows the access and debugging of any part of a Linux system.

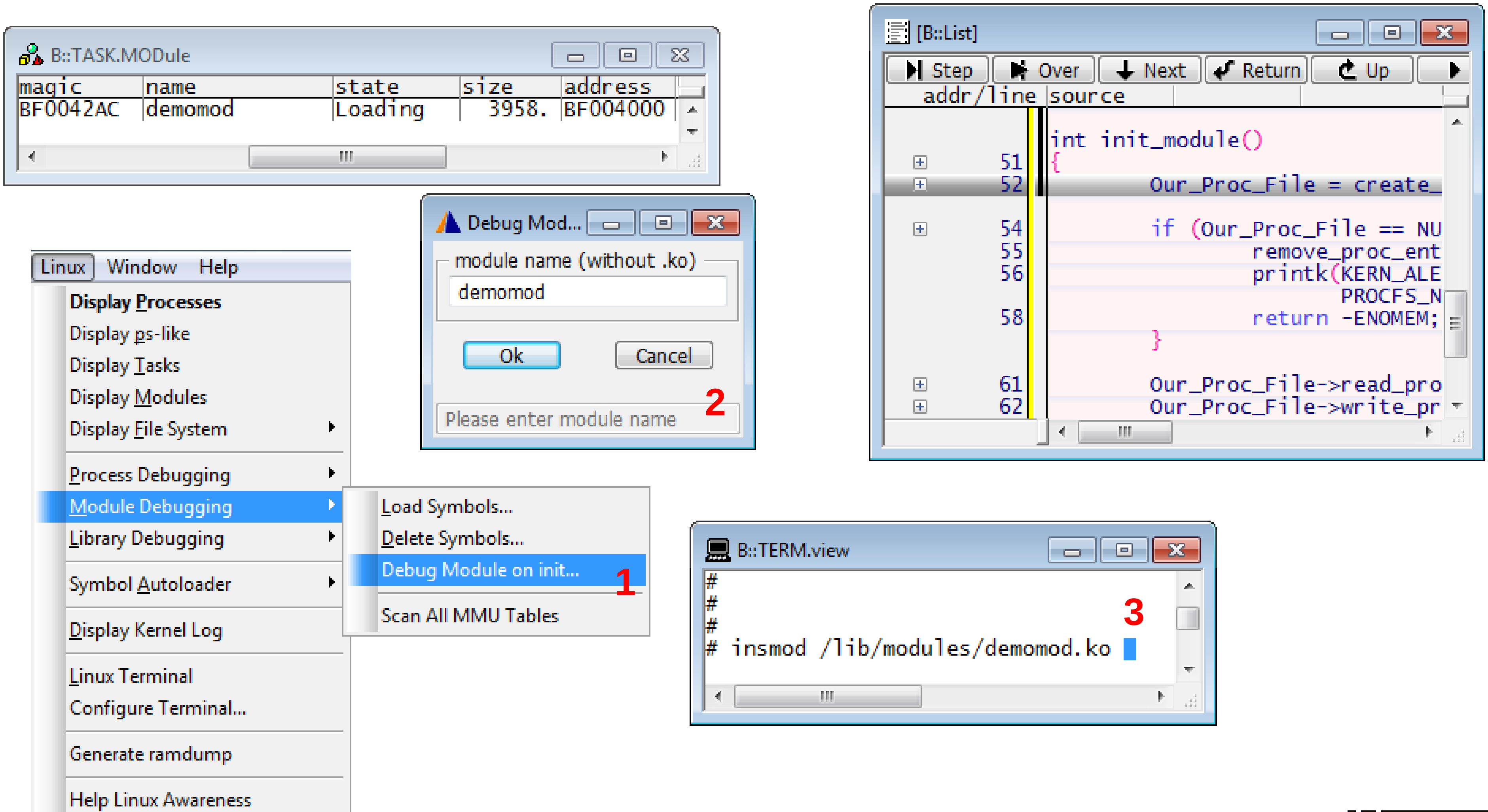


TRACE32 Linux Awareness menu

- Debugging the kernel
- Debugging kernel modules
- Debugging processes/threads
- Debugging libraries
- Automatically loading and unloading symbols for kernel modules, processes and libraries
- Display kernel information (file systems, kernel log, device tree…)

Debugging linux components: kernel module

Debug Module on init

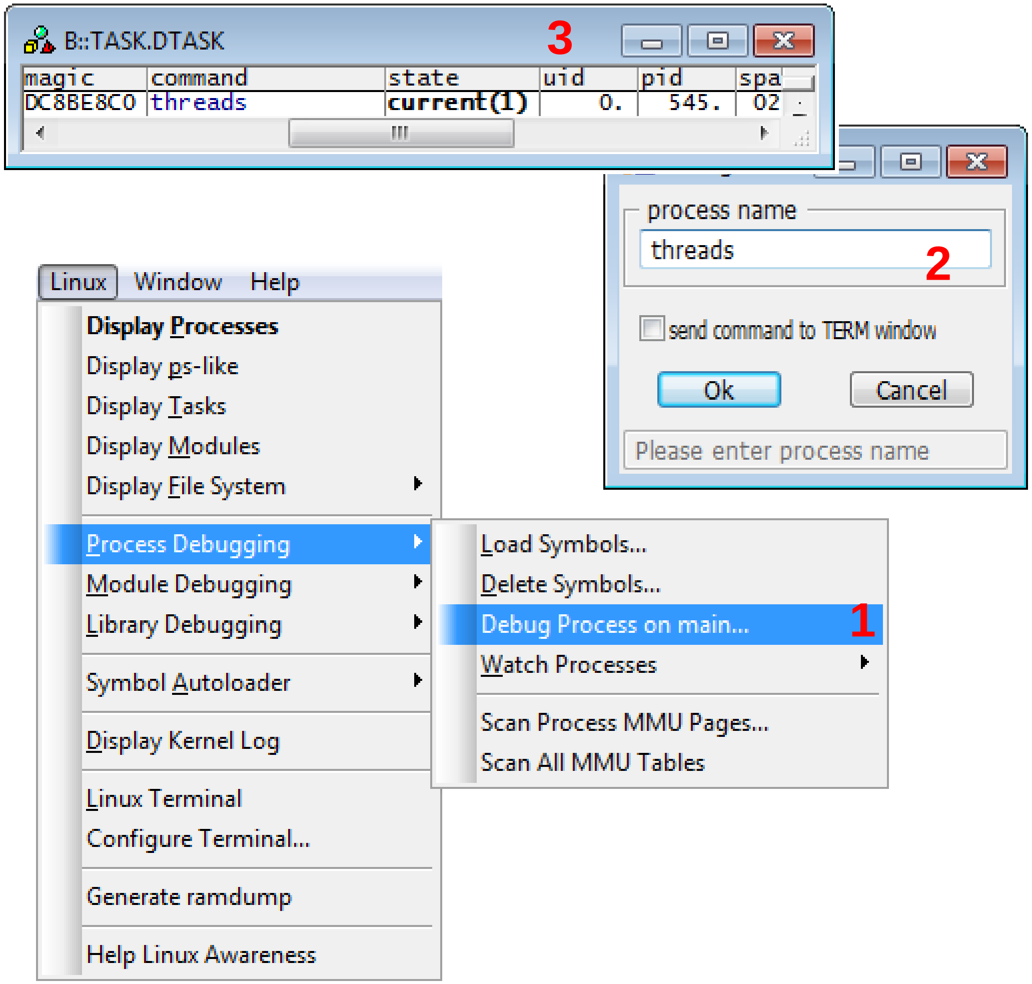


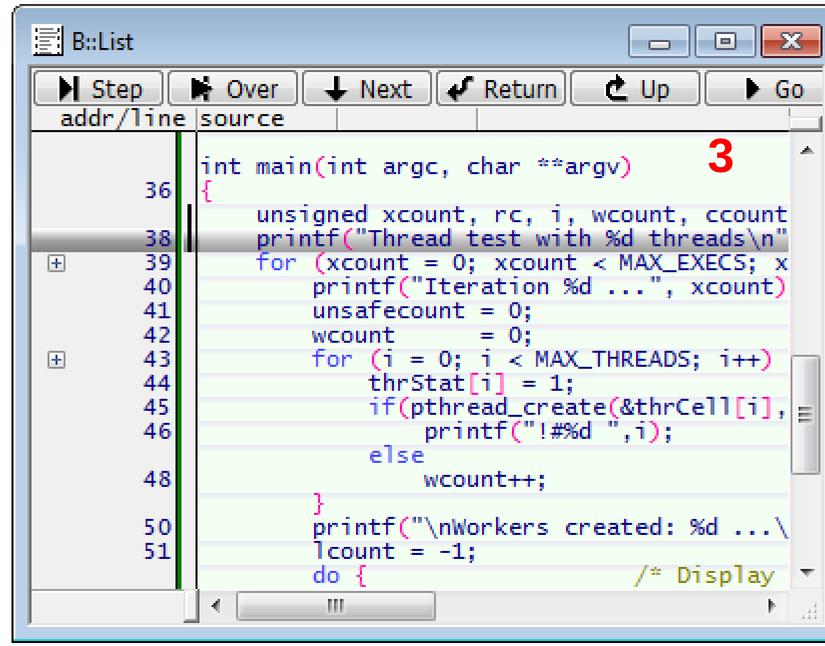
T32 DEMO

LAUTERBACH DEVELOPMENT TOOLS

Debugging linux components: process (1)

Debug Process on main





As soon as the process starts the debugger will detect it, will automatically load his symbols and will stop at the main() entry

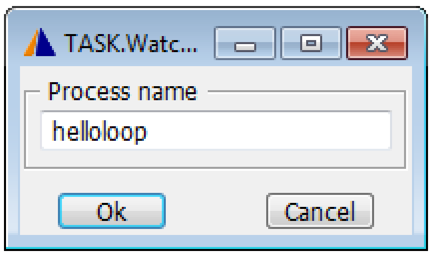
LAUTERBACH DEVELOPMENT TOOLS

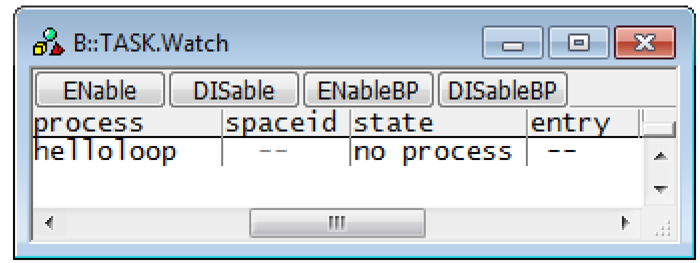
Debugging linux components: process (2)

Process Debugging – Watch Processes (autoloader)

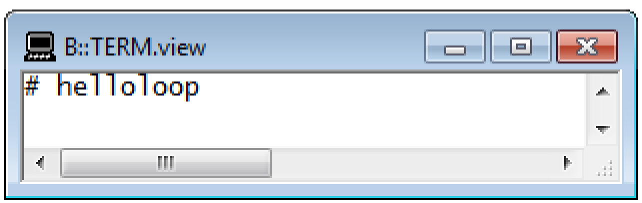
Adding a process to the "watch" list, his symbols will be automatically loaded when it start, and deleted when it exit

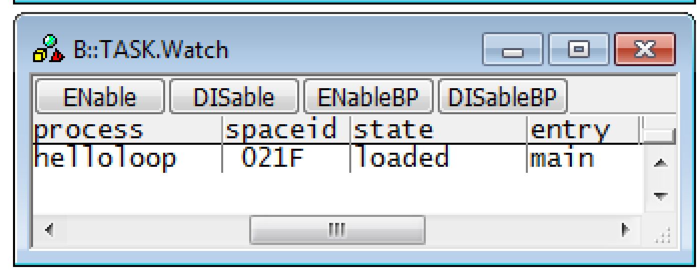
add *helloloop* to watch list



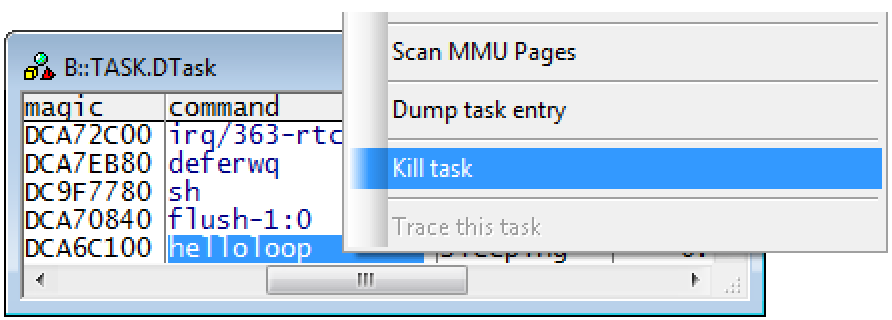


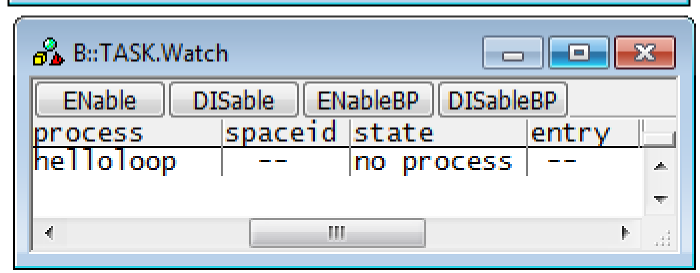
Start *helloloop*





symbols loaded



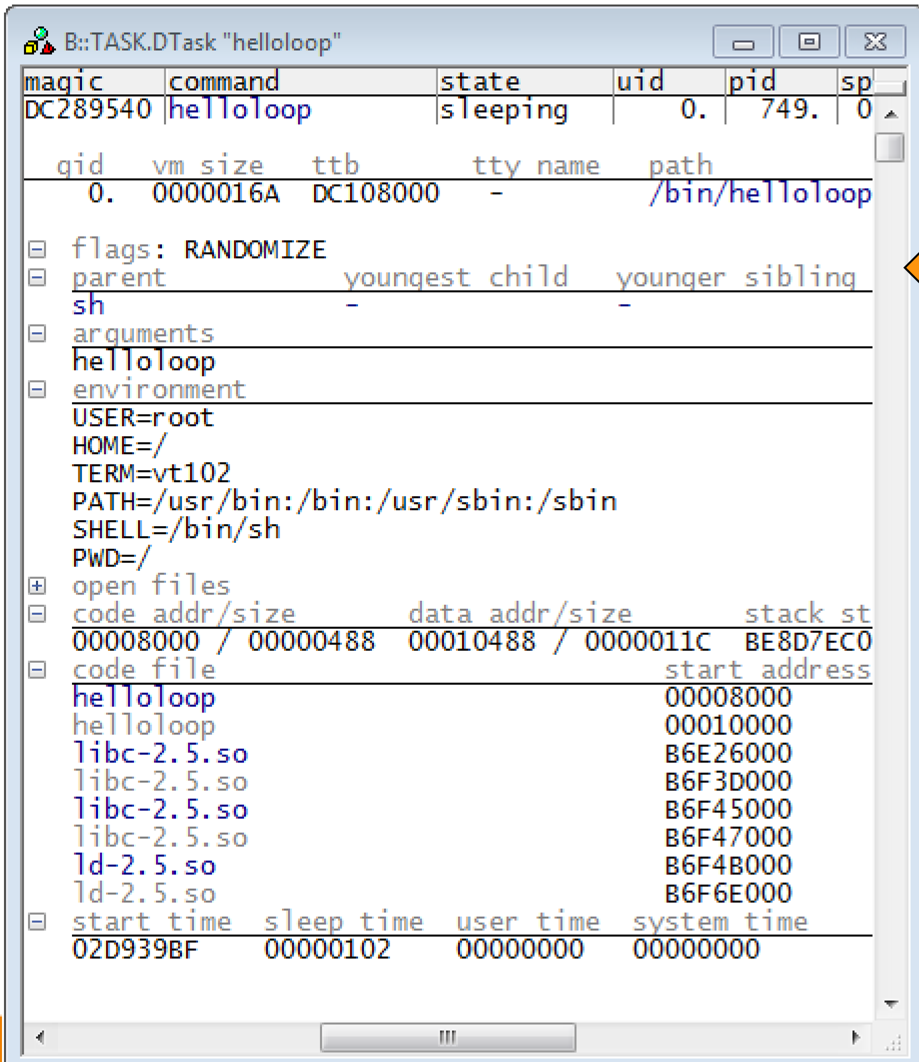


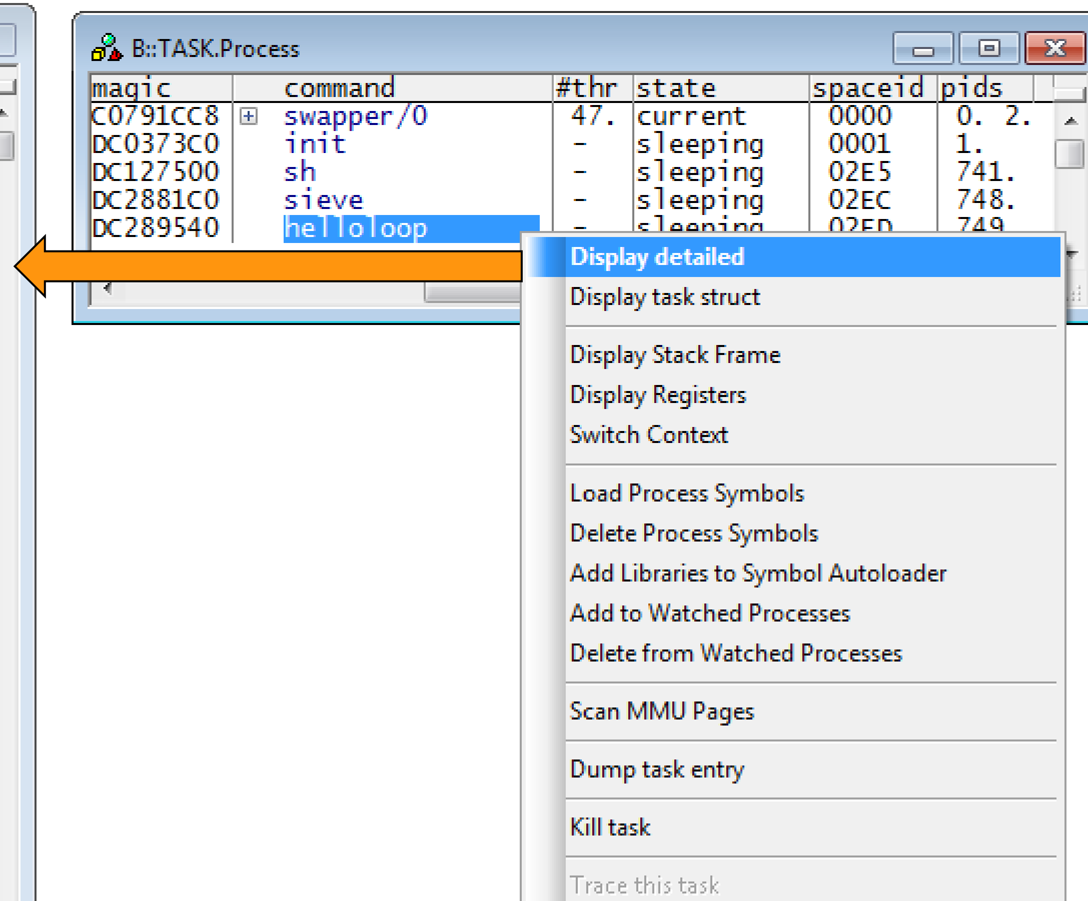
symbols deleted

LAUTERBACH DEVELOPMENT TOOLS

Debugging linux components: process (3)

TRACE32 can show detailed informations of any process or thread



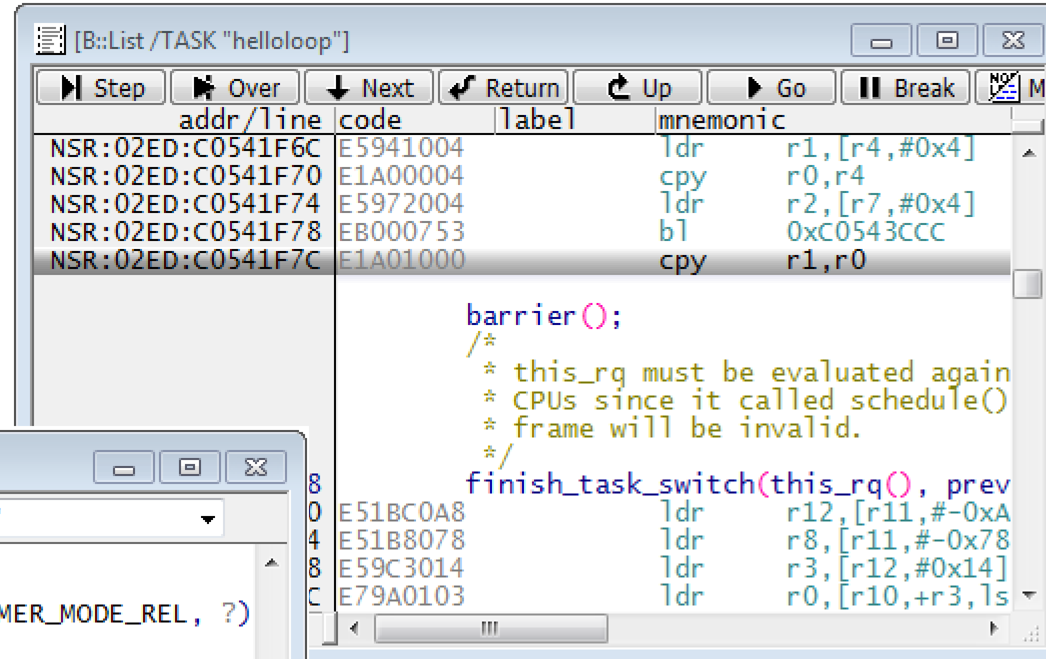


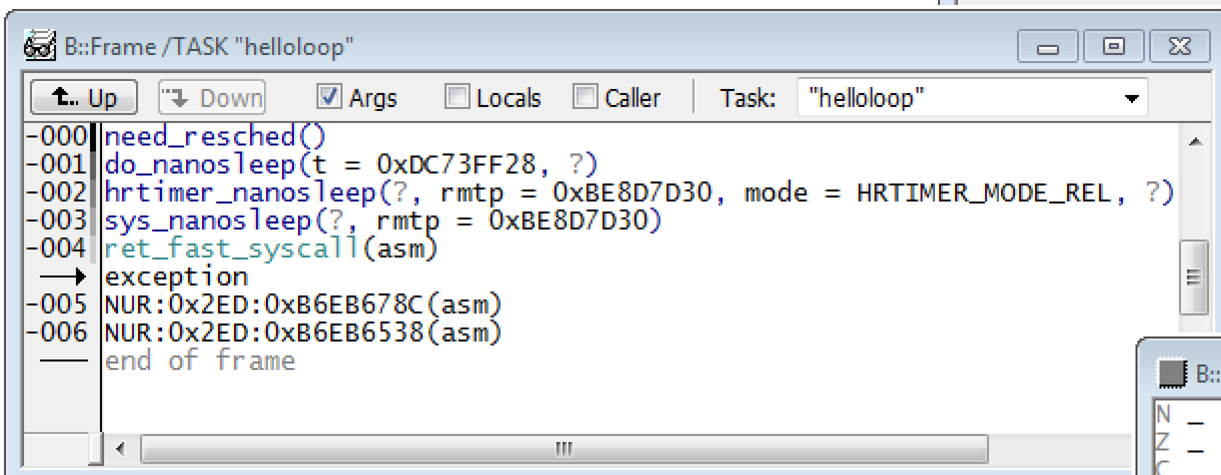
Display task specific information

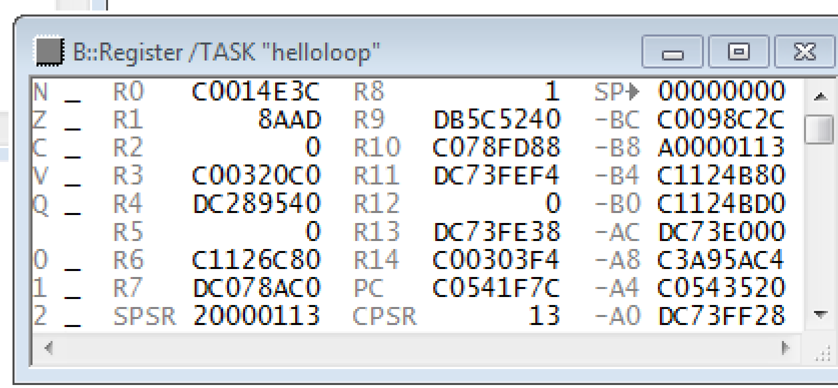
LAUTERBACH DEVELOPMENT TOOLS

Debugging linux components: process (4)

TRACE32 can show the execution context (program, data, registers) of any process or thread even if suspended, or if you are in break elsewhere

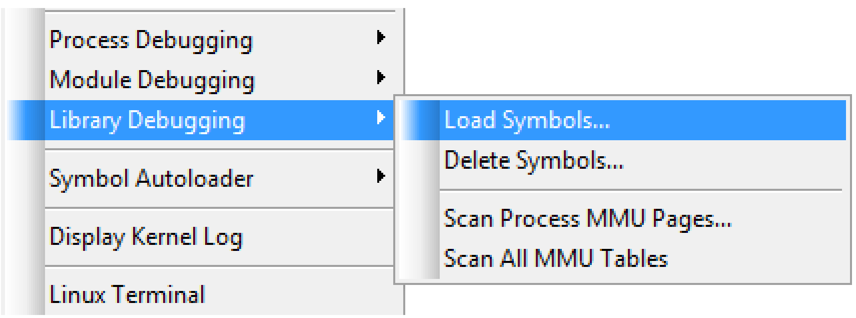


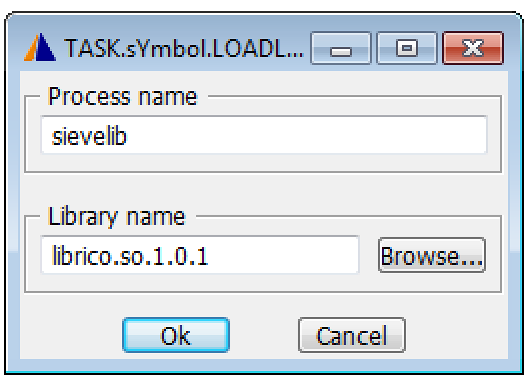


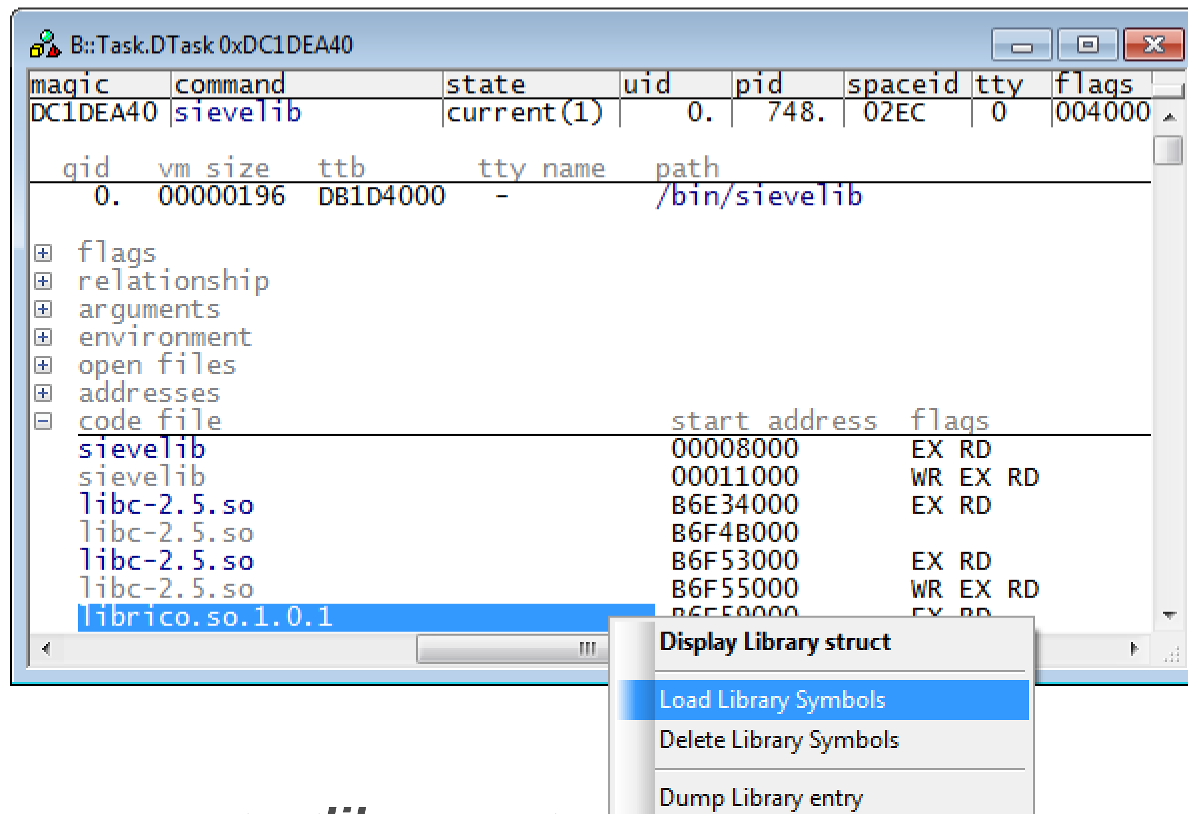


Debugging linux components: library

TRACE32 can load symbols of any library used by a process and enable its debugging







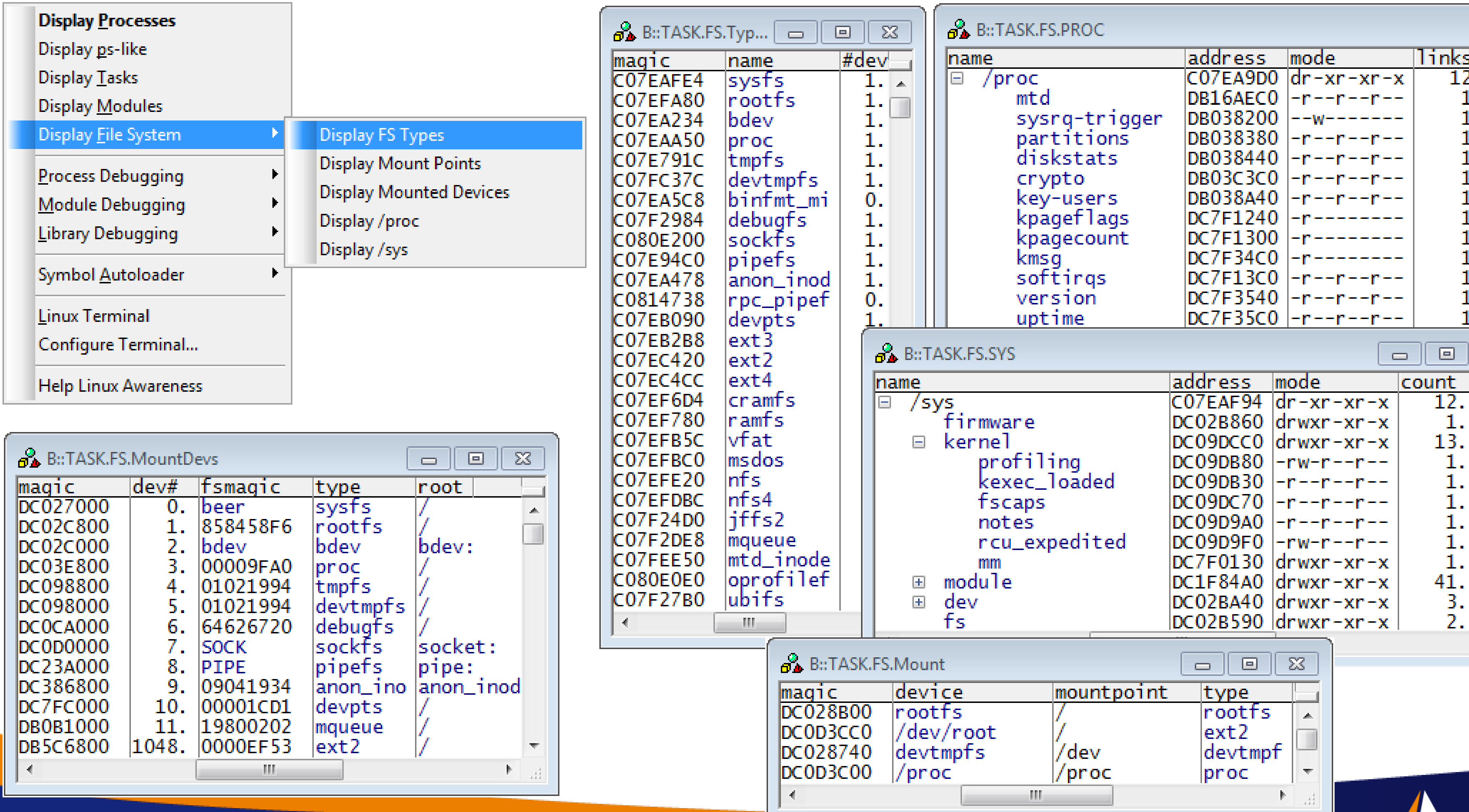
TASK.sYmbol.LOADLIB <*process_name*> <*lib_name*>

T32 DEMO

LAUTERBACH DEVELOPMENT TOOLS

Debugging linux components: file system informations

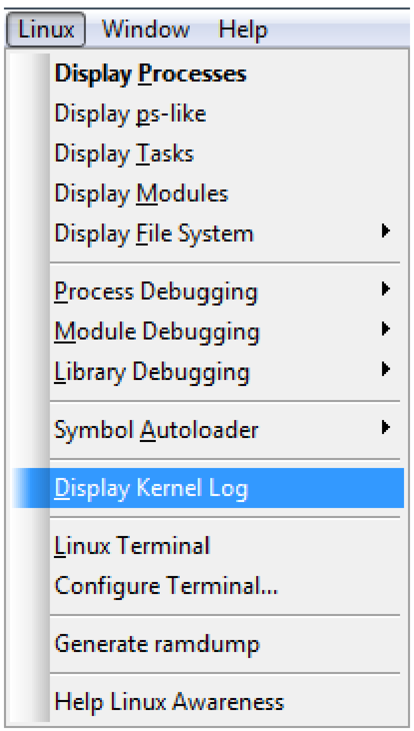
TRACE32 can show detailed information about the file systems

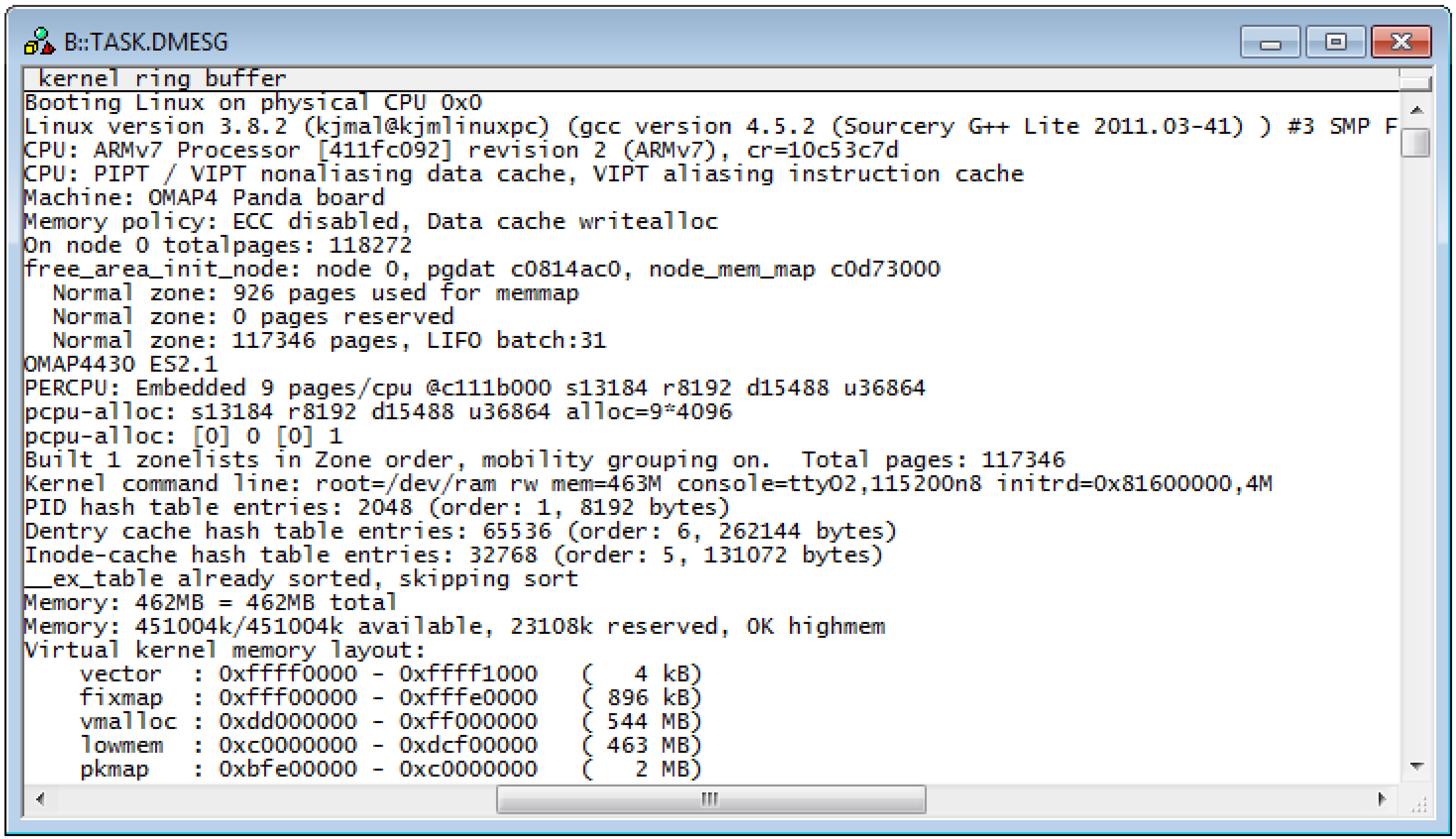


LAUTERBACH DEVELOPMENT TOOLS

Debugging linux components: kernel log

TRACE32 can show the Kernel Log by accessing directly in memory the kernel ring buffer

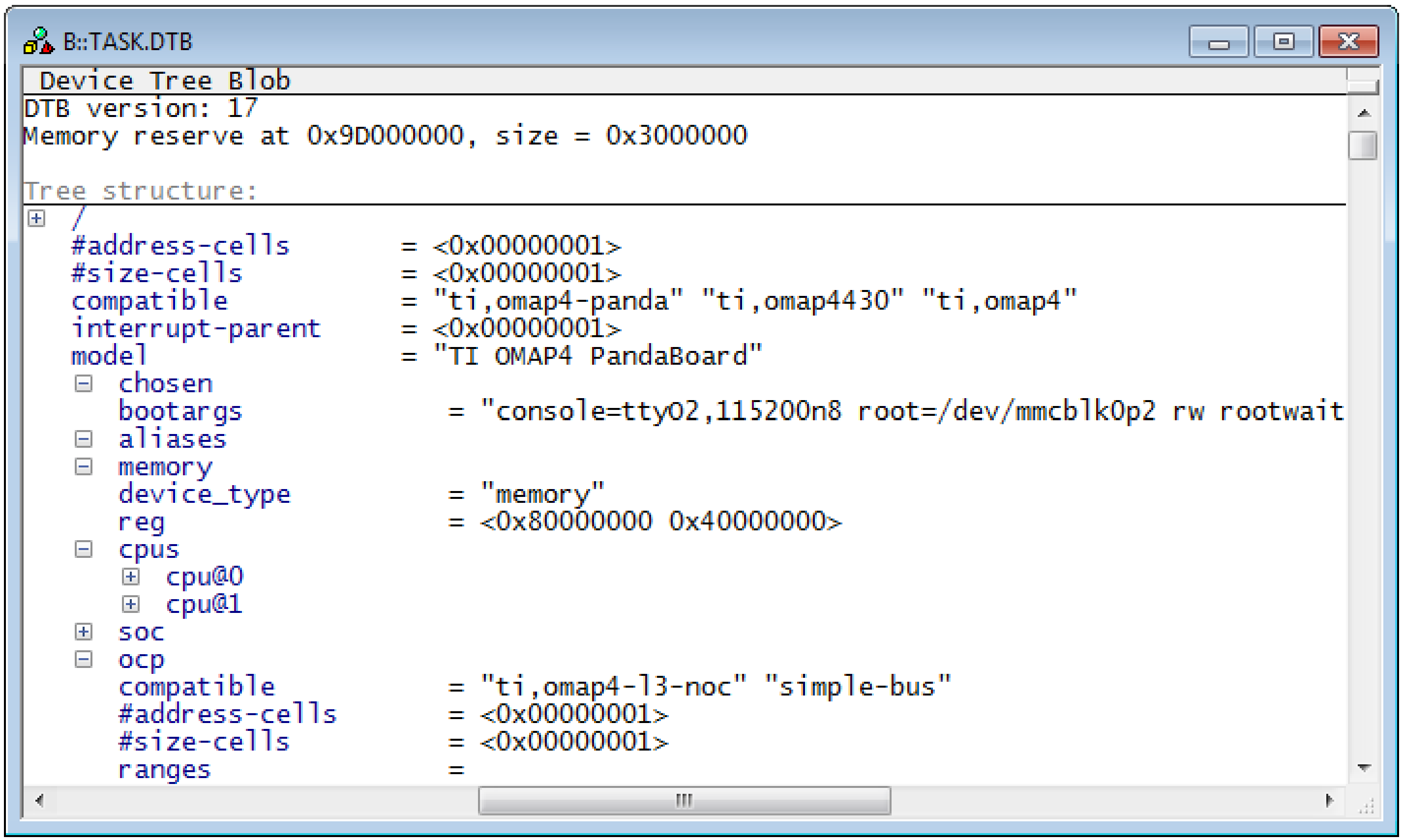




LAUTERBACH DEVELOPMENT TOOLS

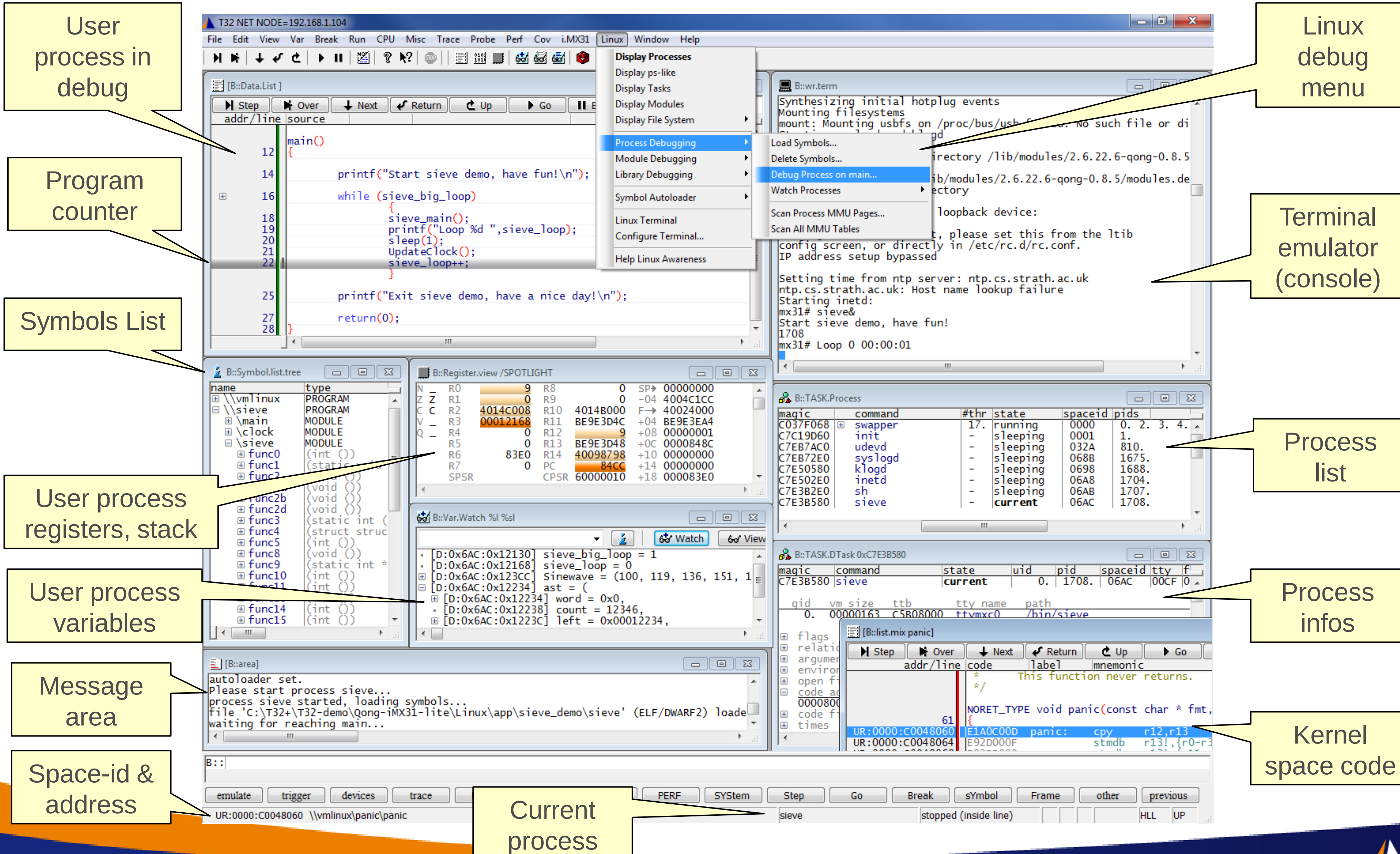
Debugging linux components: device tree

TRACE32 can load and display the Device Tree Blob (if used)



LAUTERBACH DEVELOPMENT TOOLS

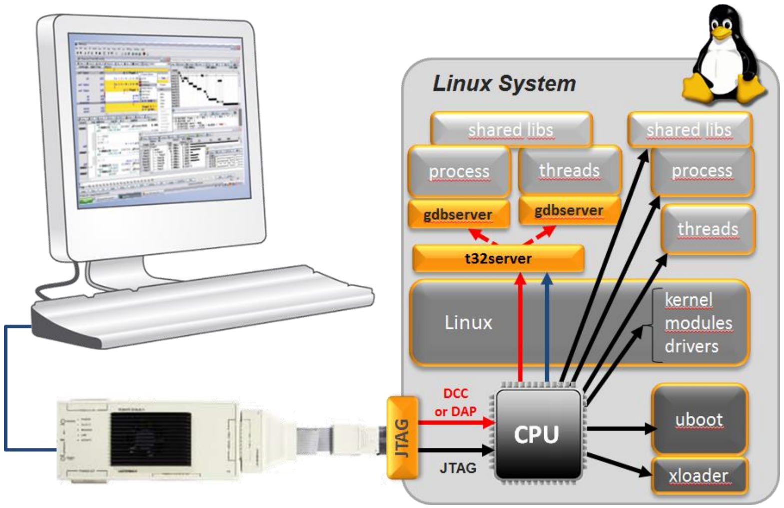
TRACE32 debugging linux



Agenda

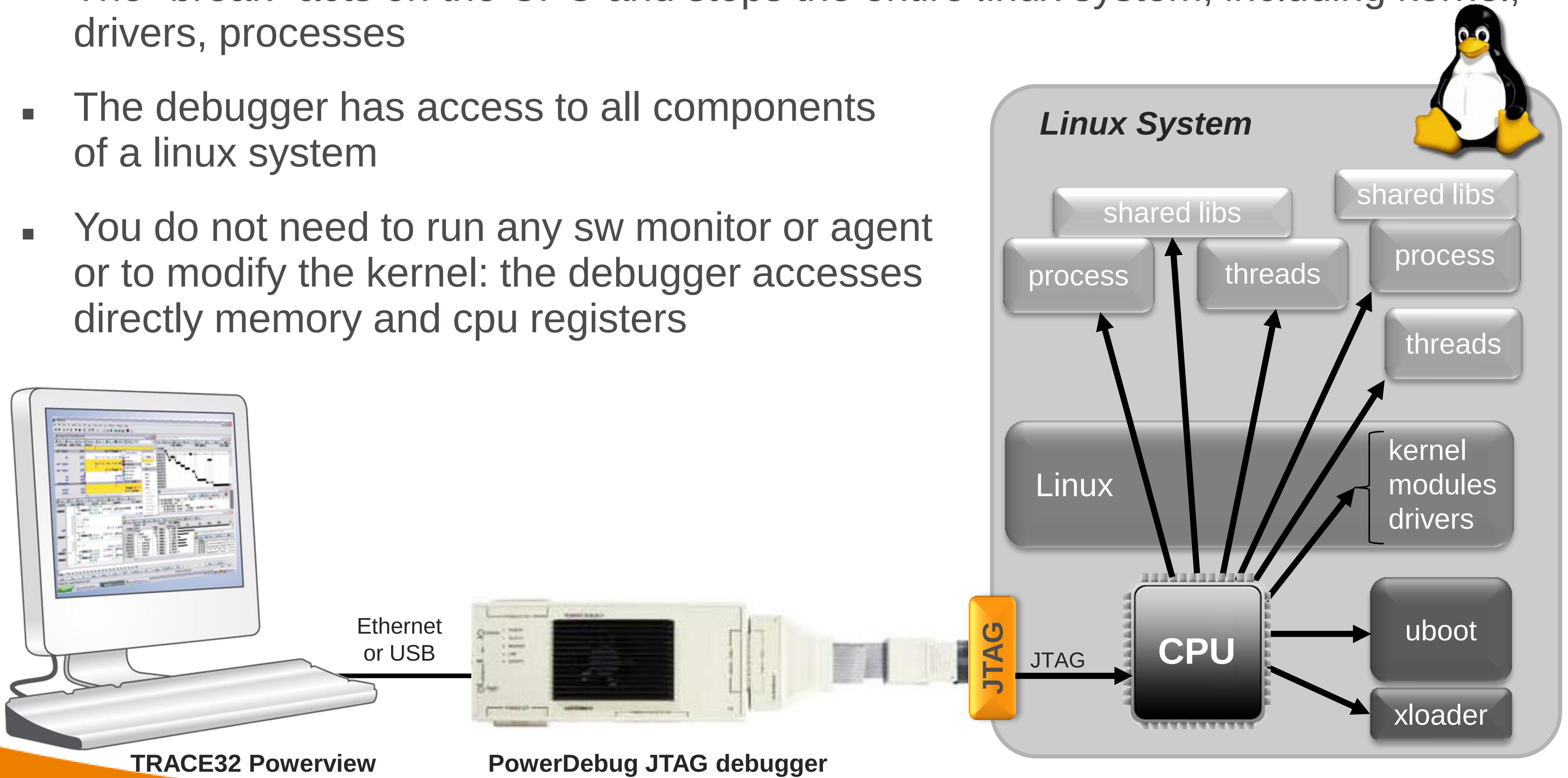
Seminar and live demo

- Linux debugging: problems & solution
- Debugging all linux components
- **Stop-mode & run-mode debugging**
- Trace, performance, profiling
- TRACE32 PowerTools
- Q&A



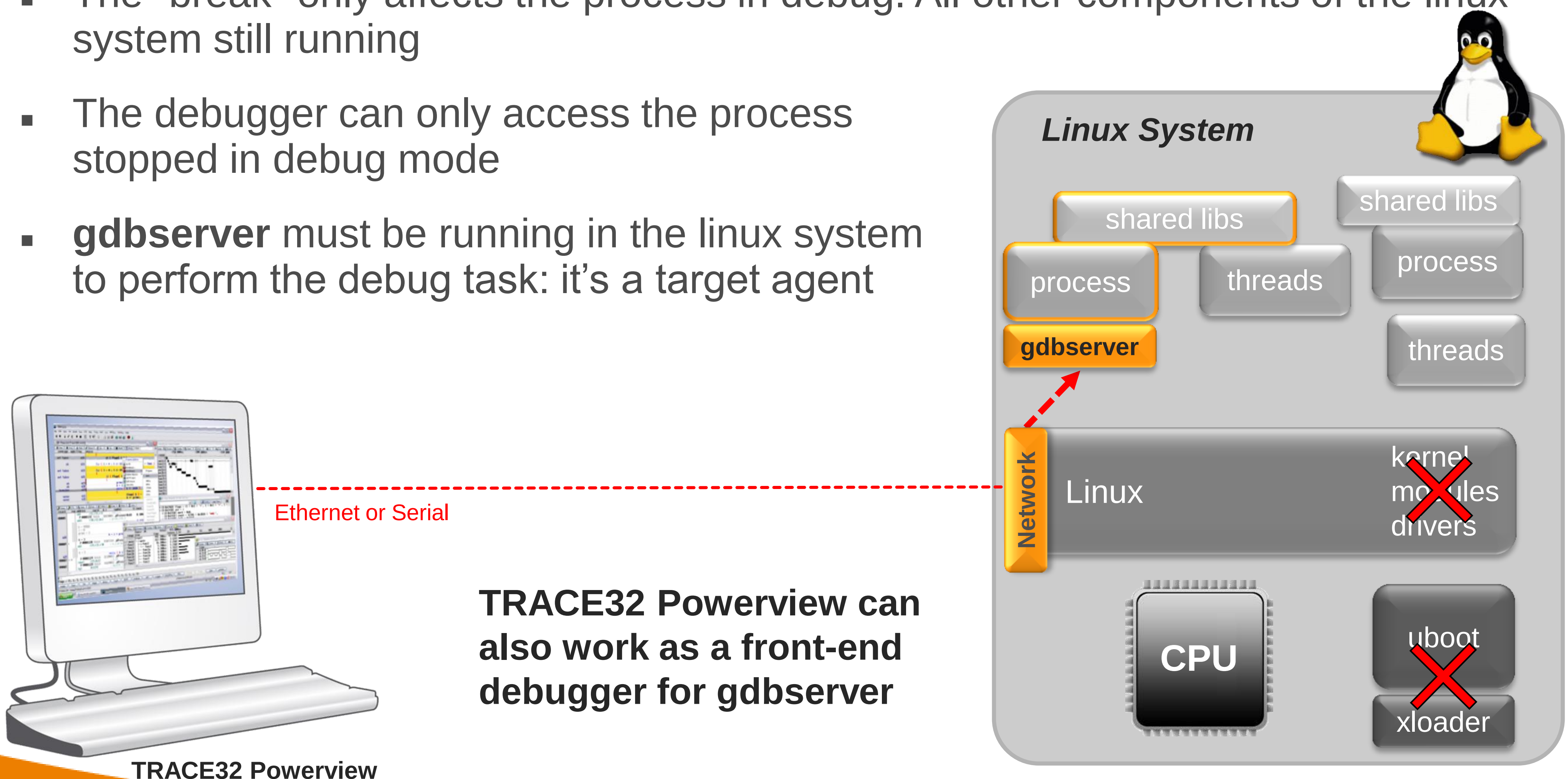
What we mean by «stop-mode debugging» ?

- Debugging via JTAG
- The "break" acts on the CPU and stops the entire linux system, including kernel, drivers, processes
- The debugger has access to all components of a linux system
- You do not need to run any sw monitor or agent or to modify the kernel: the debugger accesses directly memory and cpu registers



What we mean by «run-mode debugging» ?

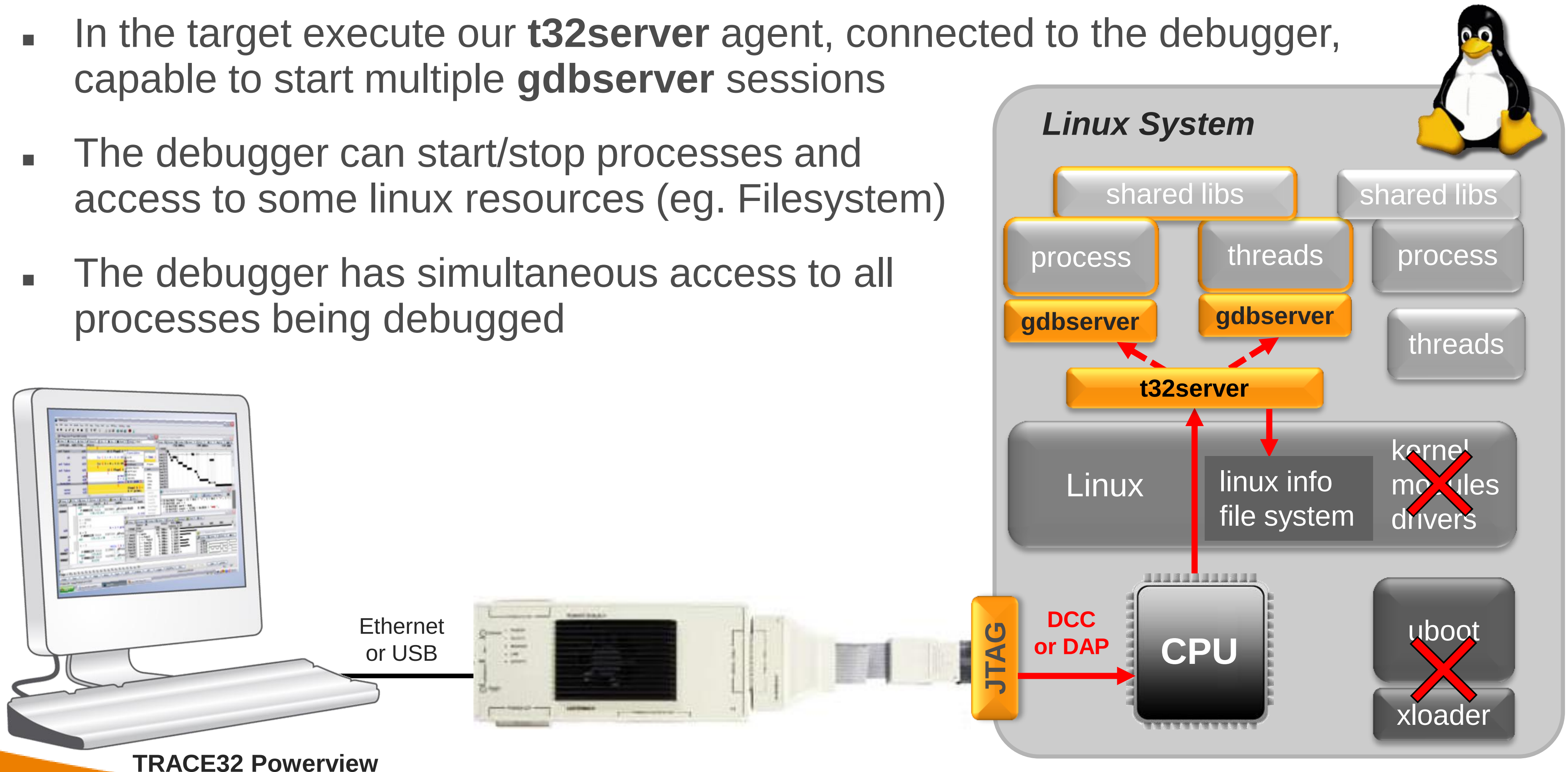
- Debugging via a communication channel: serial, ethernet
- The "break" only affects the process in debug. All other components of the linux system still running
- The debugger can only access the process stopped in debug mode
- **gdbserver** must be running in the linux system to perform the debug task: it's a target agent



TRACE32 Powerview can also work as a front-end debugger for gdbserver

Advanced «run-mode debugging»

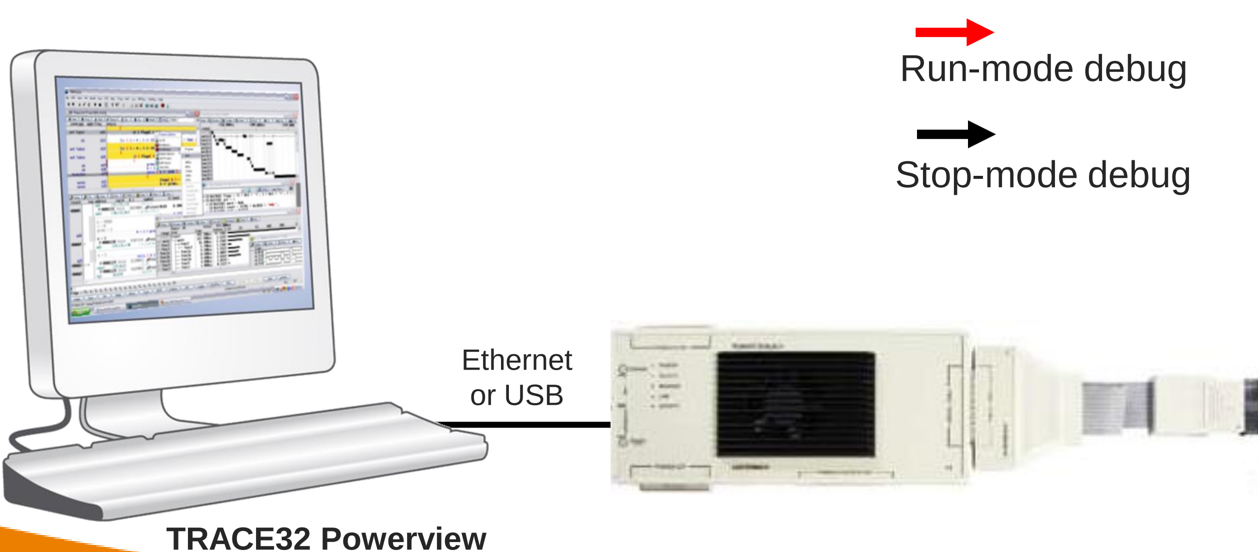
- Run-mode debugging through the communication channel JTAG DCC/DAP. Do not need ethernet or serial link, do not need drivers, JTAG only
- In the target execute our **t32server** agent, connected to the debugger, capable to start multiple **gdbserver** sessions
- The debugger can start/stop processes and access to some linux resources (eg. Filesystem)
- The debugger has simultaneous access to all processes being debugged

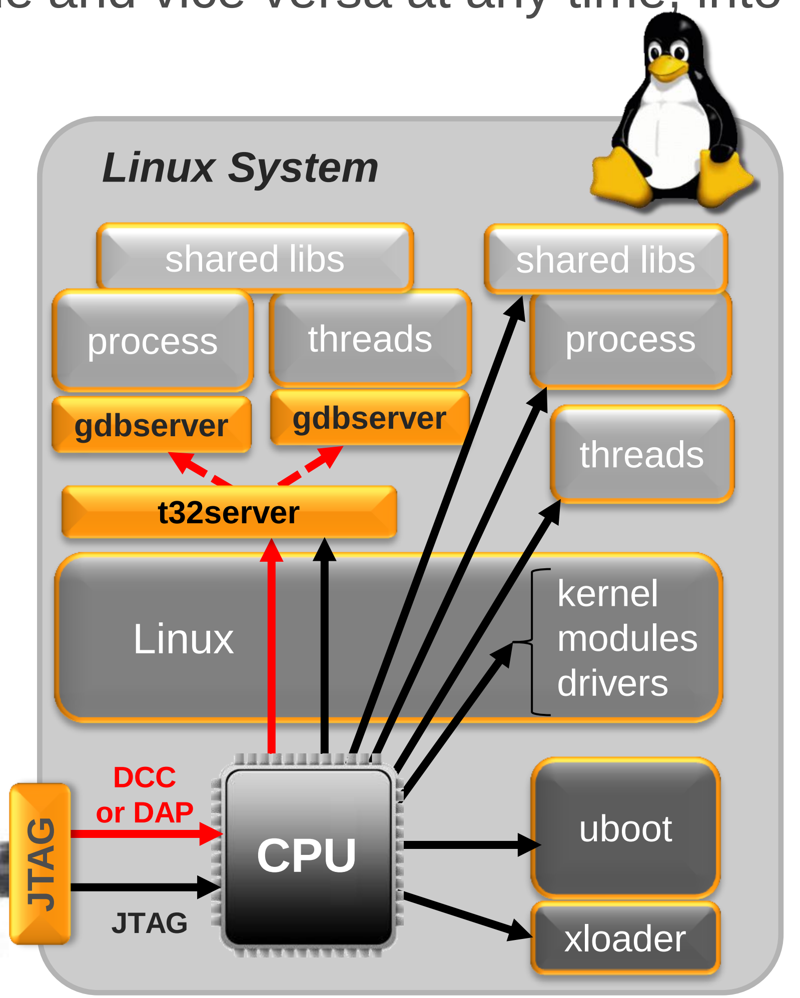


LAUTERBACH DEVELOPMENT TOOLS

Integrated stop-mode & run-mode debugging

- Integration of stop-mode debugging and run-mode debugging via JTAG DCC/DAP
- TRACE32 can switch from stop-mode to run-mode and vice versa at any time, into the same debug session
- It combines the best of both debug modes, allowing users to choose the best approach to quickly solve any bug



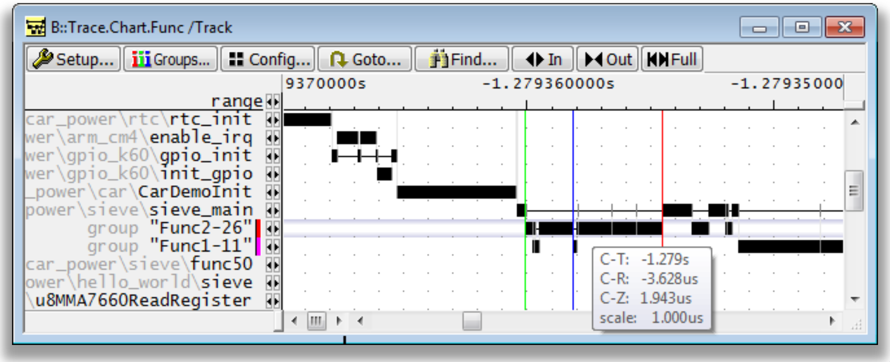


T32 DEMO

Agenda

Seminar and live demo

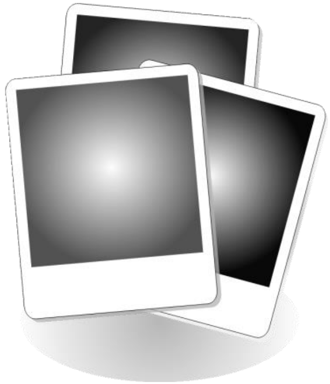
- Linux debugging: problems & solution
- Debugging all linux components
- Stop-mode & run-mode debugging
- **Trace, performance, profiling**
- TRACE32 PowerTools
- Q&A



What is the Trace?

With the term "**trace**" we mean a system for recording the sequence of instructions executed and data read/written by a CPU, without having to stop it.

Debugging



Taking Pictures

Real-Time Tracing

Recording a Video

Trace ARM, ETM & ETB

Most of the chips have a **trace-port** through which the flow trace is transmitted outside: **off-chip trace**

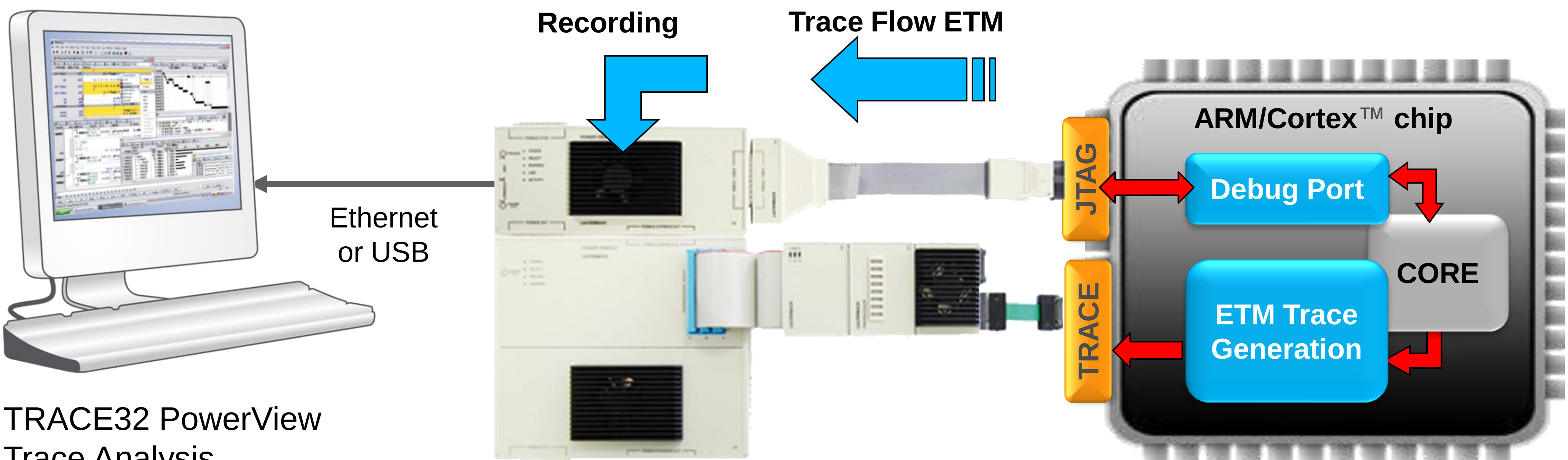
In the ARM/Cortex™ cores the off-chip trace port is called ETM (Embedded Trace Macrocell). It is a parallel or serial port at high speed. Lauterbach has made several trace-probes to support different types of trace ports.

In the case where the trace port is not available, it is possible to use a trace buffer internal to the chip called ETB (Embedded Trace Buffer). It is a **on-chip trace**, typically limited to a few KByte, for which, additional hardware is not needed.

Off-chip trace ETM: recording

While the core is running, the trace port transmits program flow and data information in a compressed way. The method has no special restrictions:

- Few pins are required
- Allows very high speeds
- Allows trigger, filtering, data trace



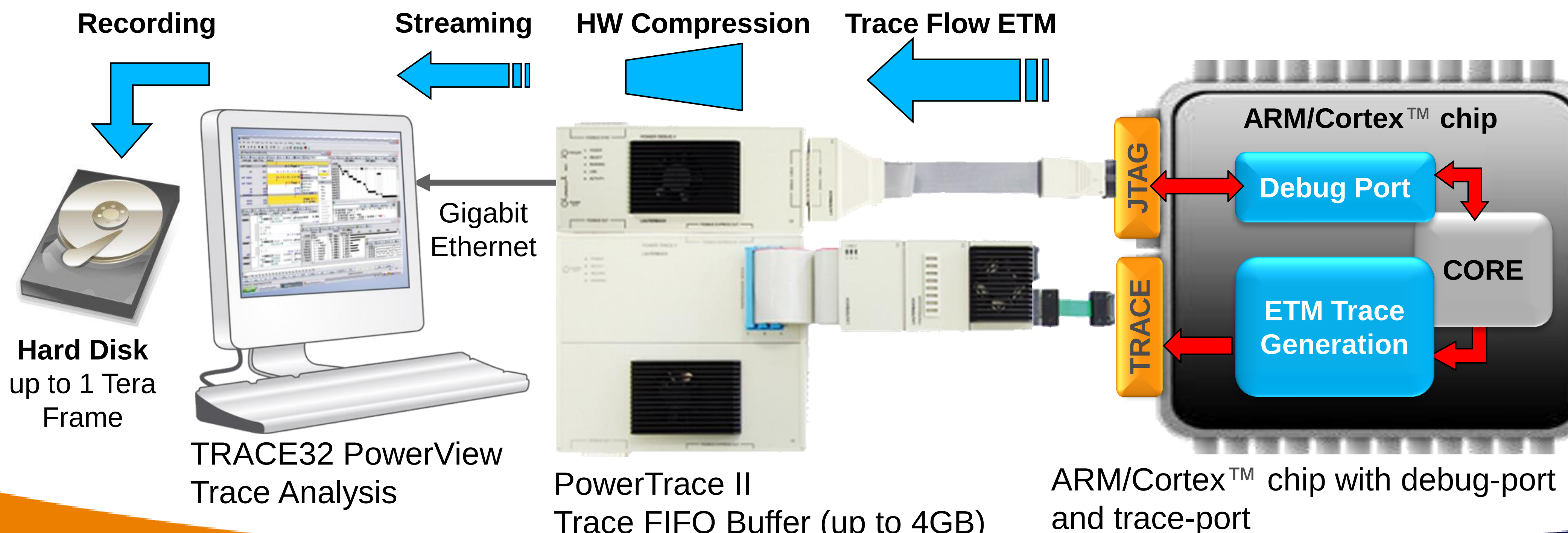
TRACE32 PowerView
Trace Analysis

PowerTrace II
Trace Storage (up to 4 Gbyte)

ARM/Cortex™ chip with debug-port and trace-port

Off-chip trace ETM: streaming

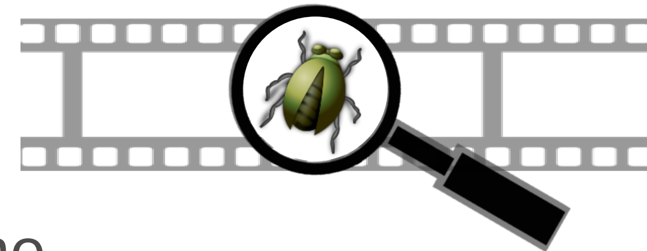
Normally, the trace is recorded inside PowerTrace which has a storage from 512MB up to 4GB. The recording time can be extended indefinitely using TRACE STREAMING. In this way, the trace-flow is compressed by PowerTrace II and transferred via gigabit ethernet to the host-PC, where it is registered.



LAUTERBACH DEVELOPMENT TOOLS

The Real-Time Trace is used for:

1) **Trace-based Debugging**



Fast debugging without stopping CPU

Finding bugs that only appear in real-time

2) **Optimization** with time measures

Analyze code performance

Analyze effect of external events

3) **Qualification**

Demonstrate compliance with real-time requirements

Check the code coverage

Real-time trace with Linux

- The transmission of the ETM trace is a hardware feature of the chip, it is non-intrusive and requires no modification to Linux

- The ETM trace transmits the logical addresses of program execution. But in Linux processes are all running at the same logical addresses

How to distinguish them?

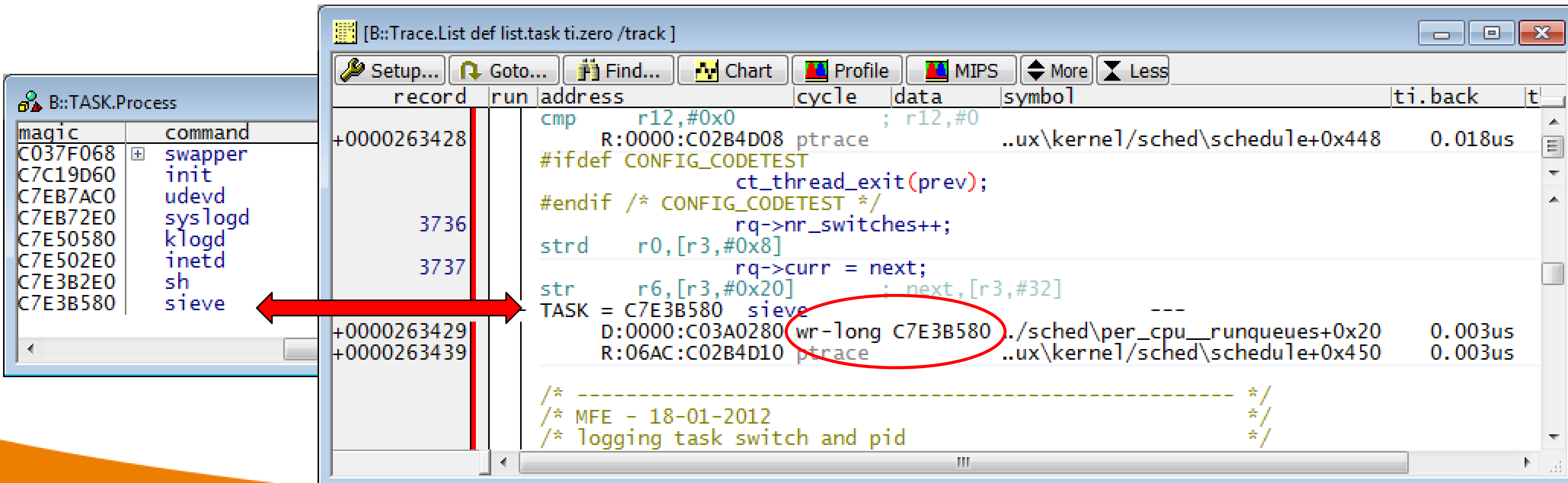
- Is necessary to capture with trace also the identifier of process switches (space-id) in order to associate the recorded code to the proper linux system component

LAUTERBACH DEVELOPMENT TOOLS

Trace with Linux: task switch

The identifier of the task switch can be easily captured by tracing writes to the variable "current process" or by tracing the "contextID" which is a ARM core register that Linux kernel updates at every task switch.

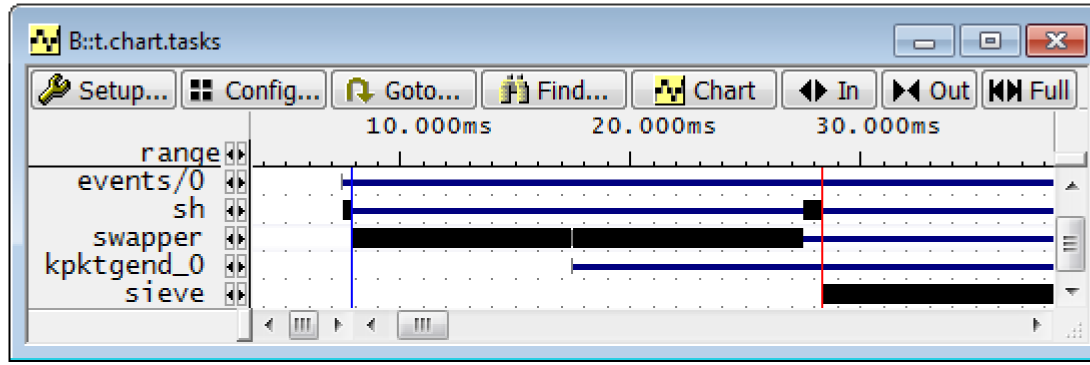
The value written (task_struct *) identifies the process and allows the immediate association of the recorded code to the proper component of the Linux system.



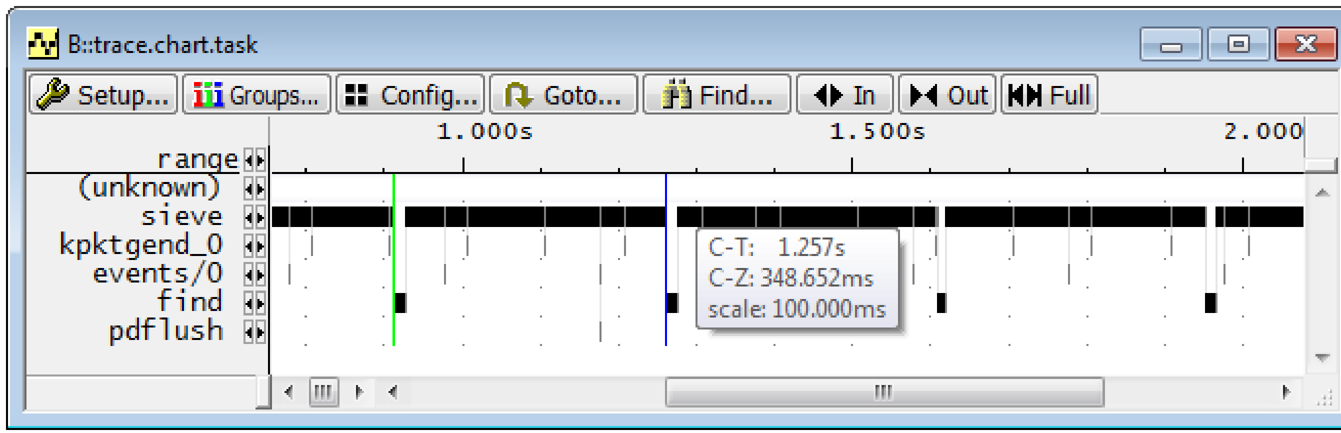
LAUTERBACH DEVELOPMENT TOOLS

Trace with Linux: task profiling

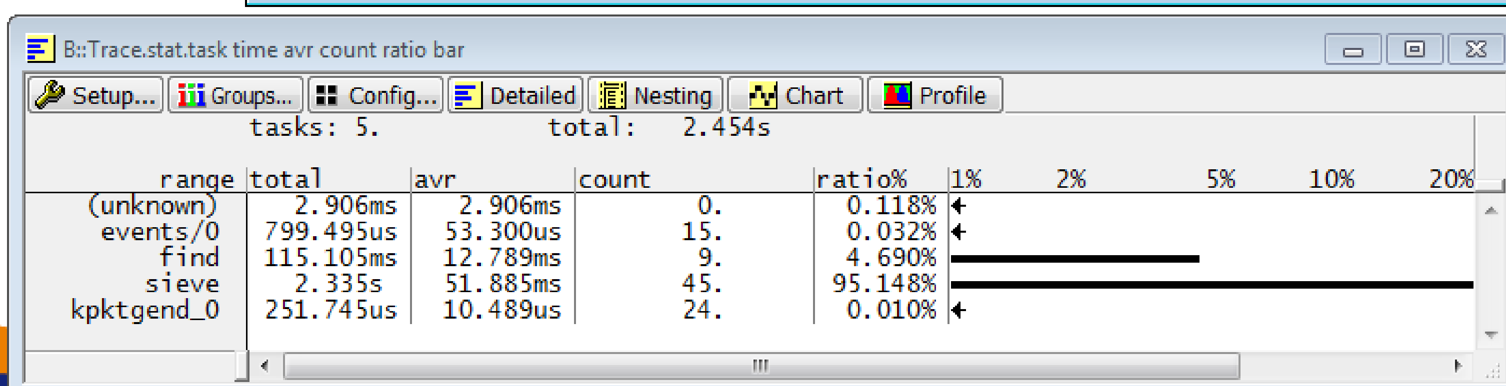
Task State
runtime chart



Task Scheduling
runtime chart



Task Timing
statistic

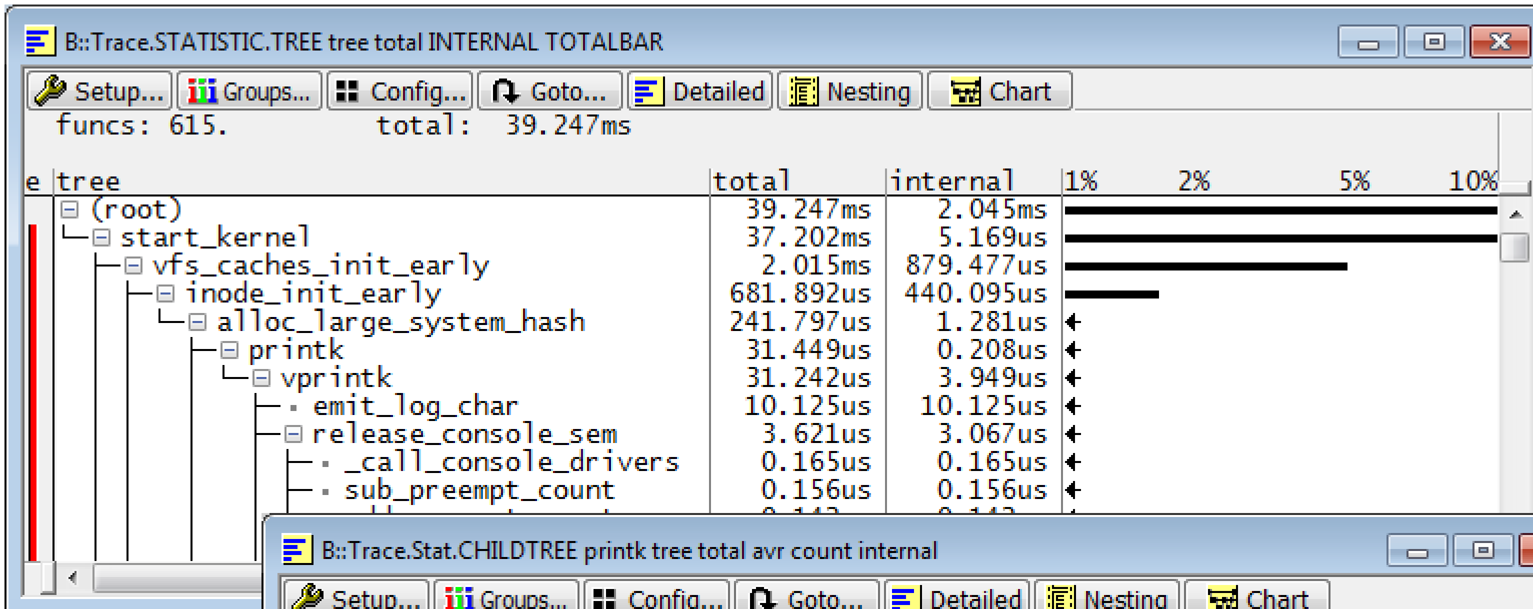


range	total	avr	count	ratio%
(unknown)	2.906ms	2.906ms	0.	0.118%
events/0	799.495us	53.300us	15.	0.032%
find	115.105ms	12.789ms	9.	4.690%
sieve	2.335s	51.885ms	45.	95.148%
kpktgend_0	251.745us	10.489us	24.	0.010%

LAUTERBACH DEVELOPMENT TOOLS

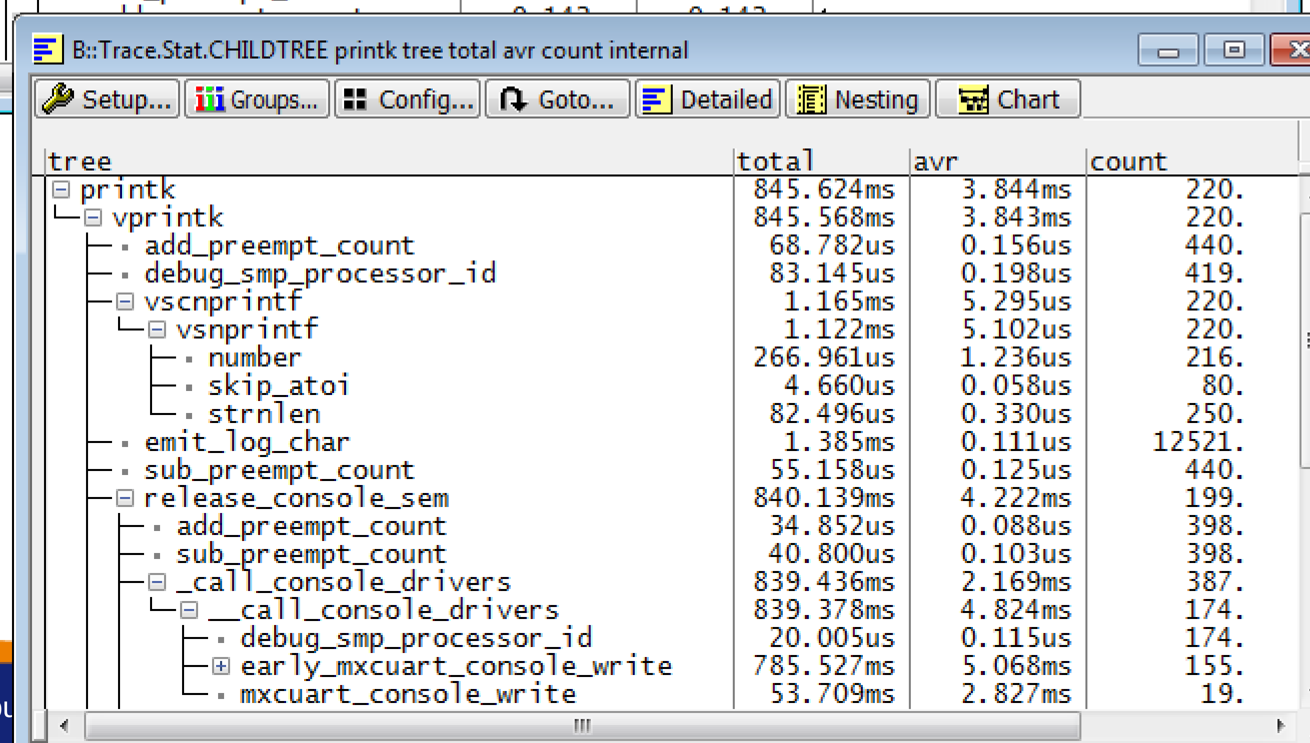
Trace with Linux: code profiling

Statistic Tree Analysis



tree	total	internal
(root)	39.247ms	2.045ms
start_kernel	37.202ms	5.169us
vfs_caches_init_early	2.015ms	879.477us
inode_init_early	681.892us	440.095us
alloc_large_system_hash	241.797us	1.281us
printk	31.449us	0.208us
vprintk	31.242us	3.949us
emit_log_char	10.125us	10.125us
release_console_sem	3.621us	3.067us
_call_console_drivers	0.165us	0.165us
sub_preempt_count	0.156us	0.156us

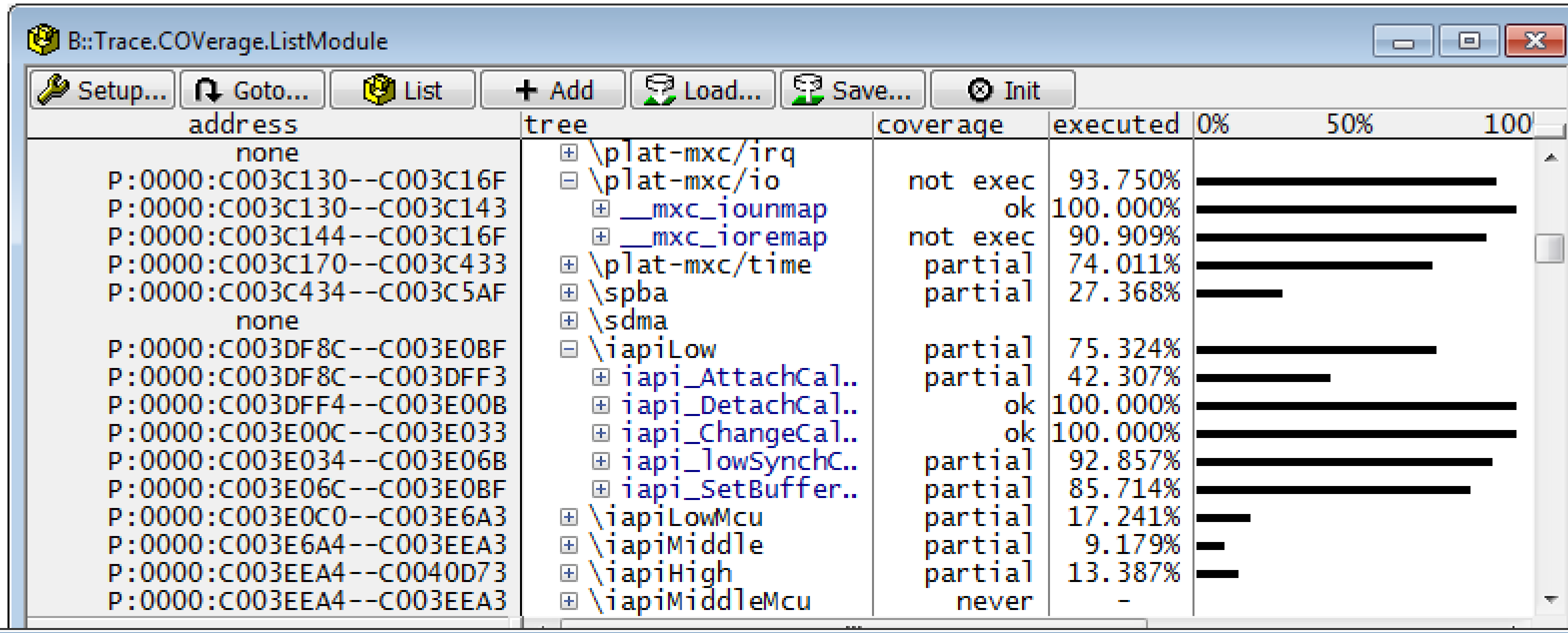
Function tree profiling of function «printk»



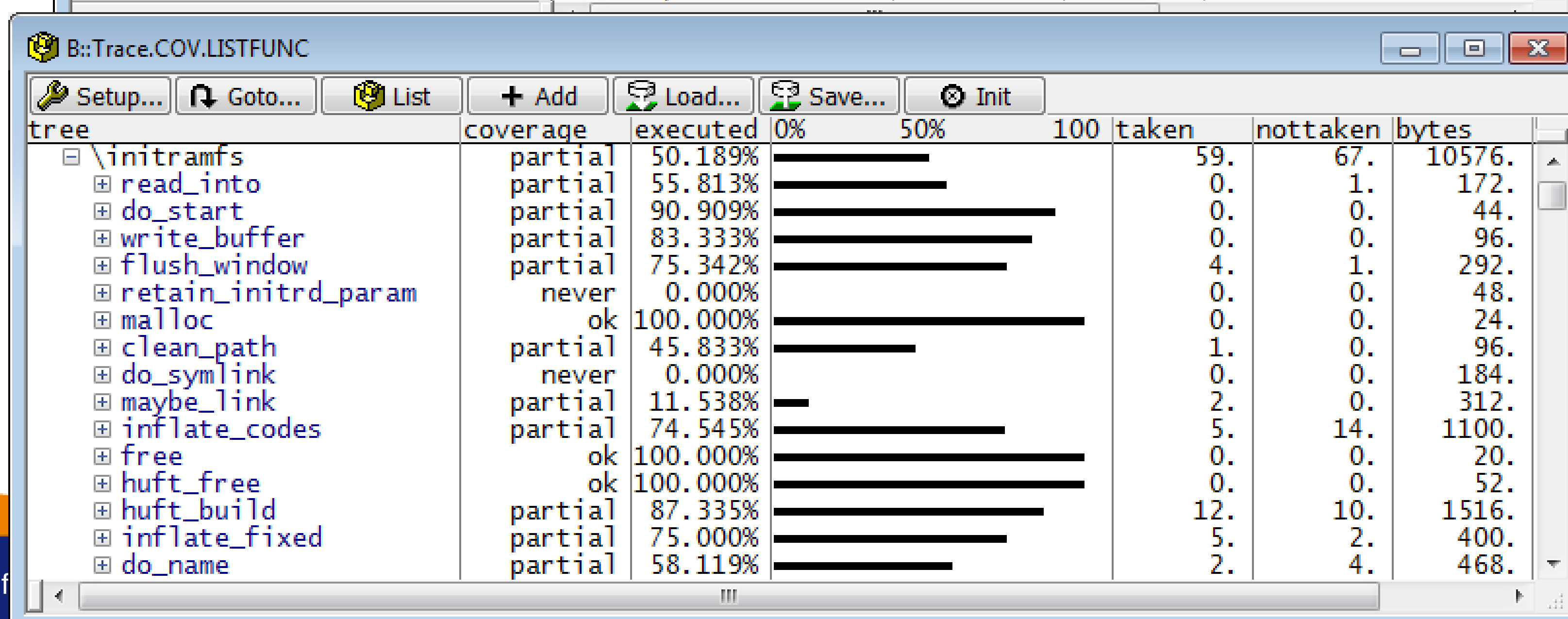
tree	total	avr	count
printk	845.624ms	3.844ms	220.
vprintk	845.568ms	3.843ms	220.
add_preempt_count	68.782us	0.156us	440.
debug_smp_processor_id	83.145us	0.198us	419.
vscnprintf	1.165ms	5.295us	220.
vsnprintf	1.122ms	5.102us	220.
number	266.961us	1.236us	216.
skip_atoi	4.660us	0.058us	80.
strnlen	82.496us	0.330us	250.
emit_log_char	1.385ms	0.111us	12521.
sub_preempt_count	55.158us	0.125us	440.
release_console_sem	840.139ms	4.222ms	199.
add_preempt_count	34.852us	0.088us	398.
sub_preempt_count	40.800us	0.103us	398.
_call_console_drivers	839.436ms	2.169ms	387.
__call_console_drivers	839.378ms	4.824ms	174.
debug_smp_processor_id	20.005us	0.115us	174.
early_mxcuart_console_write	785.527ms	5.068ms	155.
mxcuart_console_write	53.709ms	2.827ms	19.

Trace with Linux: code coverage (1)

Code coverage by object files

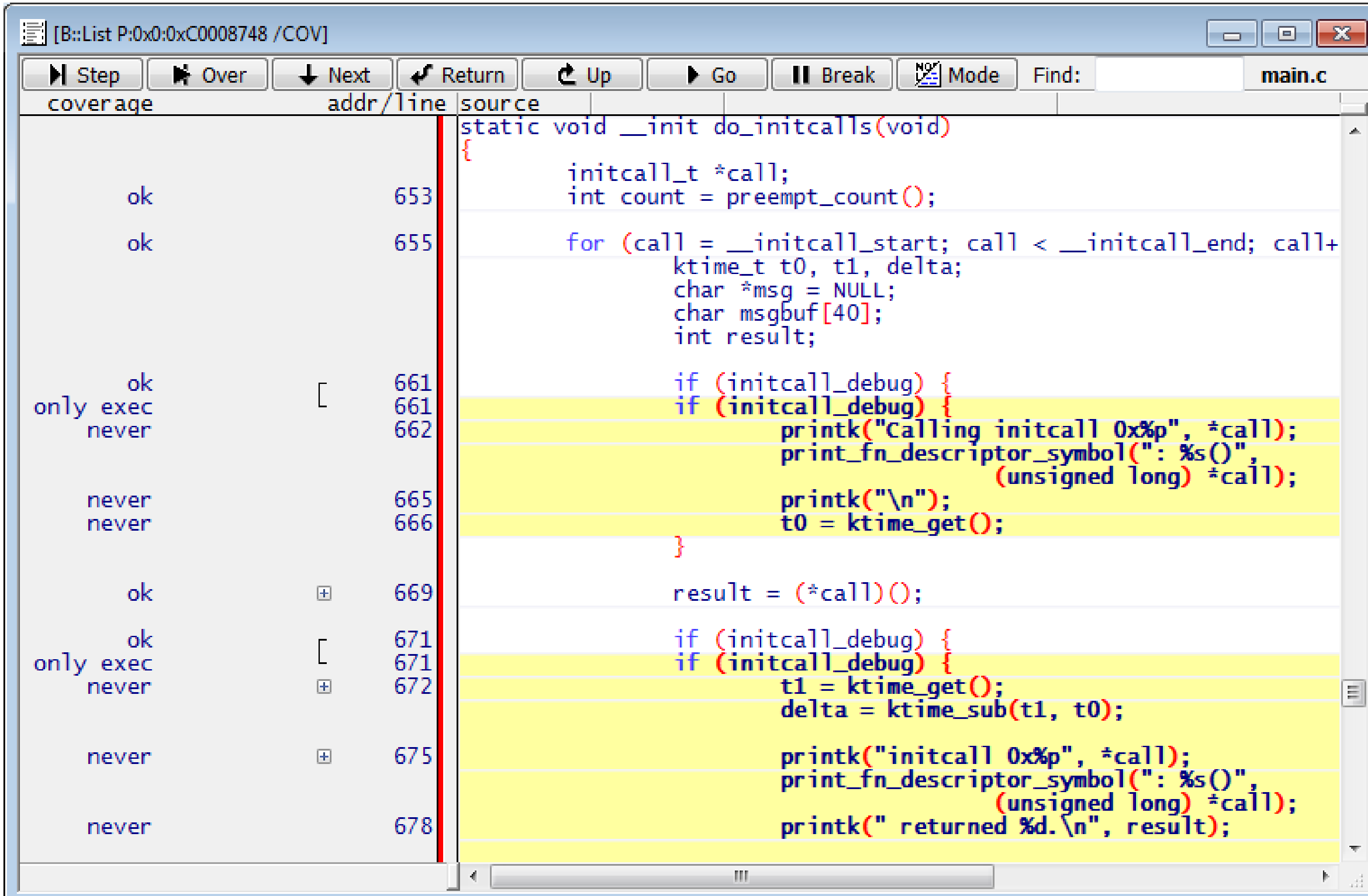


Code coverage by functions



Trace with Linux: code coverage (2)

Code coverage source code level

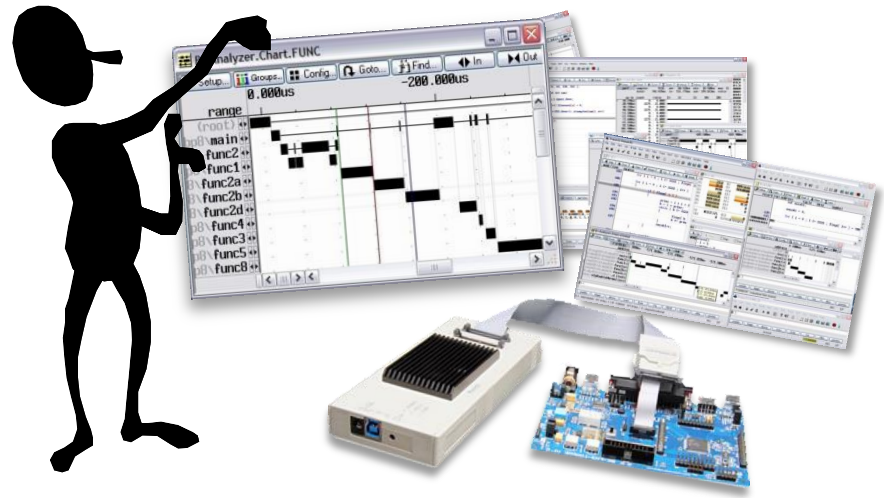


T32 DEMO ??

The trace is an important choice

The choice of trace method depends mainly on the CPU being used and its resources.

The results obtained depend on the quality of the trace tool.



The Trace is…

➔ The tool that allows you to "see" what really happens during the execution of your application.

The Trace should be considered as…

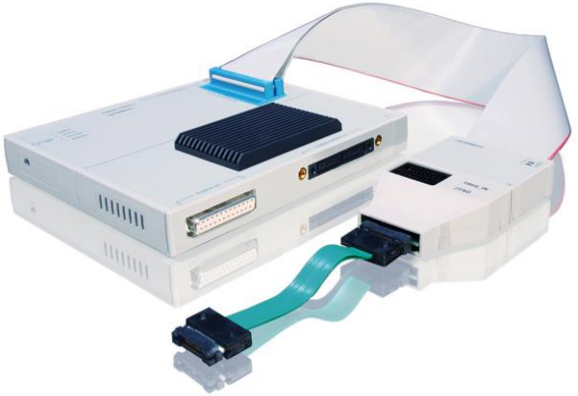
➔ The instrument to reduce development time and the best guarantee to quickly find and resolve bugs

LAUTERBACH DEVELOPMENT TOOLS

Agenda

Seminar and live demo

- Linux debugging: problems & solution
- Debugging all linux components
- Stop-mode & run-mode debugging
- Trace, performance, profiling
- **TRACE32 PowerTools**
- Q&A



Lauterbach PowerTools

Lauterbach is the world leader for debug and trace tools, with over 30 years of experience. **TRACE32 PowerTools** are the most advanced hw/sw debugger available today. It is a universal and modular hardware system that support debug-port and trace-port of many different cpu and architectures.

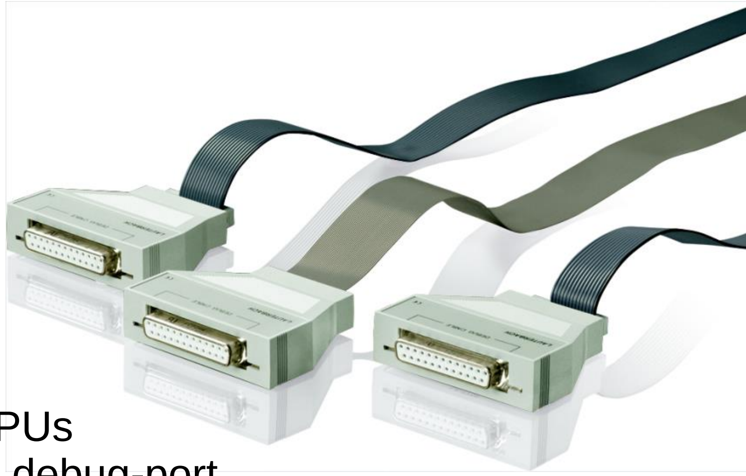
PowerDebug
(debug)

PowerTrace
(debug+trace)

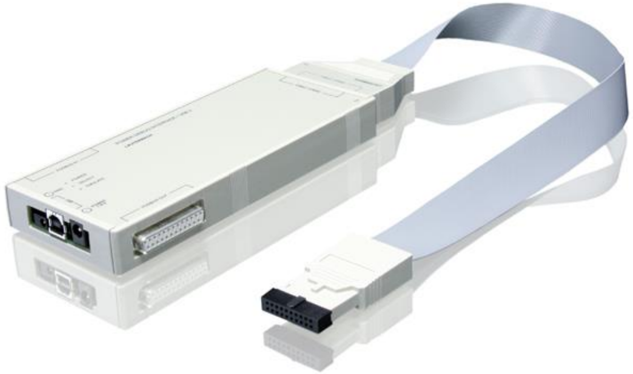
PowerIntegrator
(debug+trace+logic analyzer)

In Circuit Debuggers

A debugging system based on modular **PowerDebug** units connected to debug cables specific to different architectures and **debug ports**



Debug Cables
- Support for all CPUs
- Support for each debug-port
- Active probes at high speed
- Compatible with all PowerDebug units



PowerDebug USB-3
- Entry level system
- Link USB2/USB3

PowerDebug ETH
- Standard System
- Link USB + Eth 10/100 mbps
- Upgradable to PowerTrace

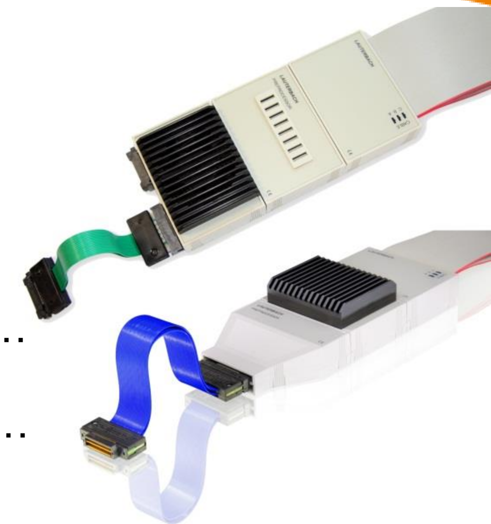
PowerDebug II
- New generation system
  Link USB + Eth 10/100/1000 mbps
- Upgradable to PowerTrace II

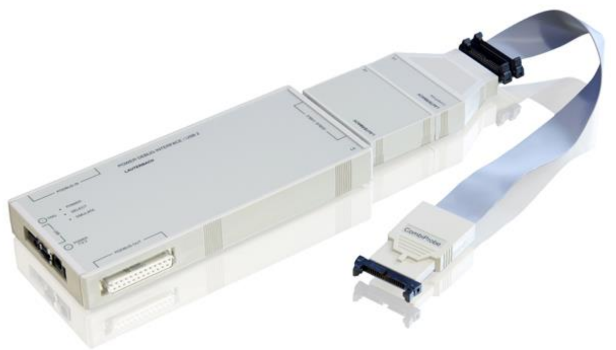
In Circuit Trace

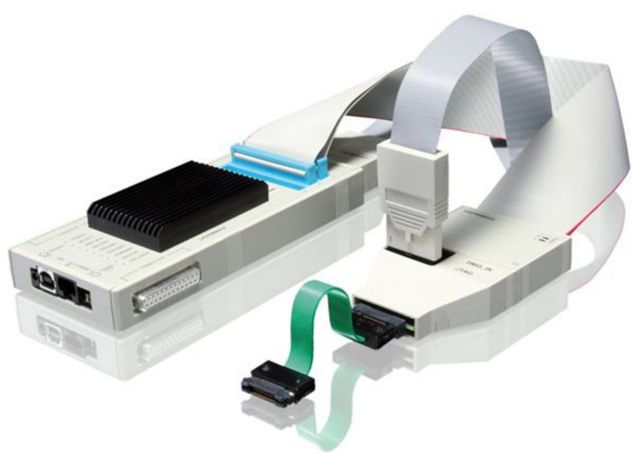
A debug+trace system based on modular **PowerTrace** units that connect debug cables and trace probes specific for different architectures and different **trace-port**

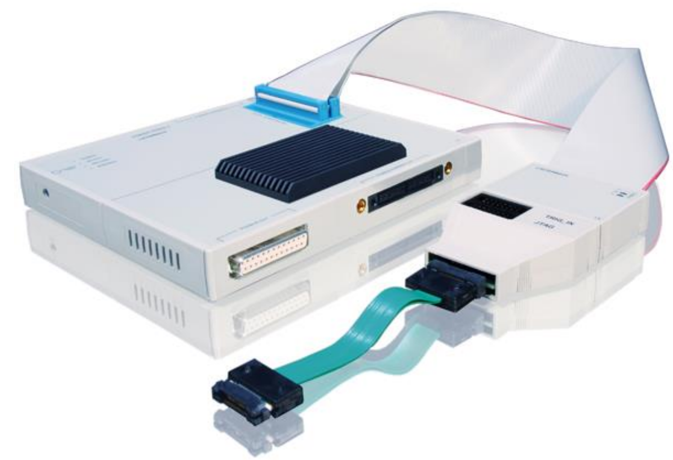
Trace Probes Autofocus

- Parallel trace ETM/NEXUS, …
- Serial Trace HSTP Aurora, …









PowerDebug Combiprobe

- Low-cost system
- 128MB trace storage
- 200 Mhz max trace clock
- 1-4 bit trace port

PowerTrace

- First generation system
- 512MB trace storage
- Up to 350 Mhz trace clock

PowerTrace II

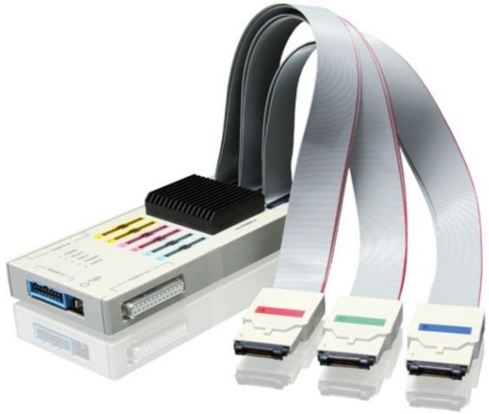
- New generation system
- 1/2/4 GB trace storage
- > GHz trace clock (HSTP)
- Trace Streaming

Logic Analyzers

Any PowerDebug and PowerTrace can be greatly enhanced with the addition of a integrated logic/protocol analyzer: **PowerIntegrator.**

A PowerIntegrator can be used for:

- **I/O timing & trigger**
- **Protocol analyzer**
  CAN, FlexRay, LIN, SPI, USB, I2C, Jtag, Seriale, PCI, DigRF, …
- **Data logger**
- **Energy test**
- **Bus-trace**
  for cpu without trace port



PowerIntegrator
- 512 K-Sample
- Max 204 channels
- Max 500Mhz

Probes
- Digital and analog
- For protocols
- For memory bus

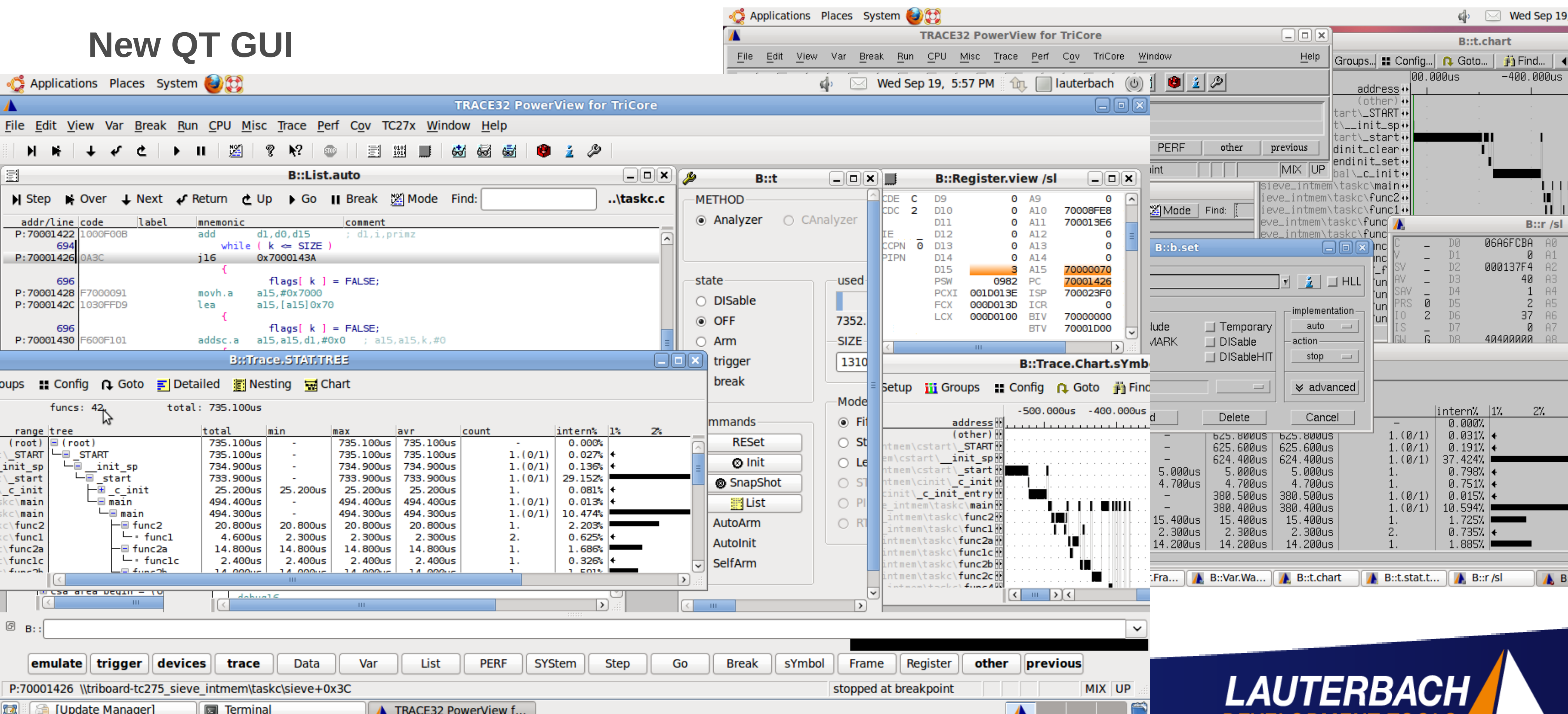
PowerIntegrator II
- Max 256000 K-Sample
- Max 102 channels
- Max 500 Mhz
- Stimuli Generator

TRACE32 PowerView for linux/QT

TRACE32 PowerView is available for Windows, MacOS-X, and Linux and Workstations. Is now available a new version of PowerView GUI for linux QT. Both the new QT version and old Motif version are available on TRACE32 software DVD.

Old Motif Gui

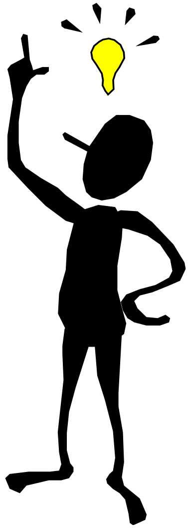
New QT GUI



Agenda

Seminar and live demo

- Linux debugging: problems & solution
- Debugging all linux components
- Stop-mode & run-mode debugging
- Trace, performance, profiling
- TRACE32 PowerTools
- **Q&A**



To learn more:

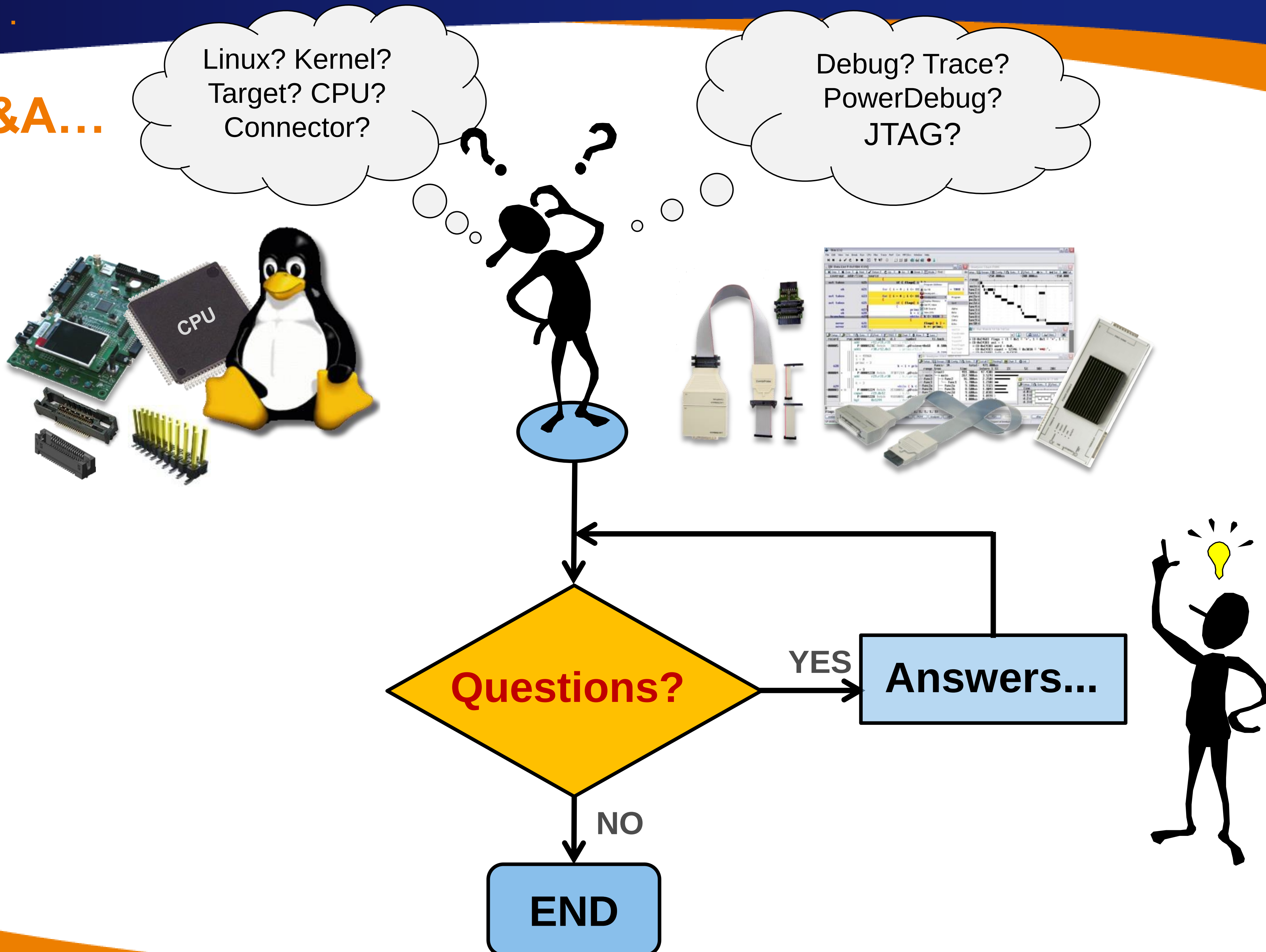
Flyers

- ✓ Debug & Trace for ARM
- ✓ Product Overview
- ✓ Linux Flyer *Advanced Debugging and Tracing tools for ARM architectures and Linux kernels*

Web

- ✓ Linux Training (training manual)
  www.lauterbach.com/pdf/training_rtos_linux.pdf
- ✓ RTOS Debugger for Linux (manual)
  www.lauterbach.com/doc/rtoslinux.pdf
- ✓ TRACE32 Startup Script (repository)
  www.lauterbach.com/scripts.html
- ✓ Linux Debugging Reference Card
  www.lauterbach.com/linux_card1_web.pdf

Q&A…


Linux? Kernel?
Target? CPU?
Connector?
Debug? Trace?
PowerDebug?
JTAG?
CPU
Questions?
YES
Answers…
NO
END

Thank you for partecipating to the seminar: the most complete tool for embedded linux debugging

Maurizio Menegotto
maurizio.menegotto@lauterbach.it

Contact for Italy:

Lauterbach SRL
tel: 02-45490282
web: www.lauterbach.it
email: info_it@lauterbach.it

International contact:

Lauterbach GmbH
web: www.lauterbach.com
email: info@lauterbach.com

LAUTERBACH DEVELOPMENT TOOLS