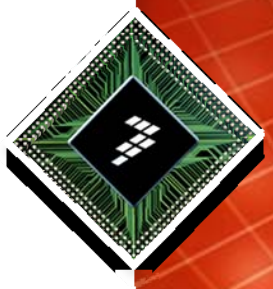
Debugging tool functionality for Multicore real-time applications under Linux

What key customers request
What works well in practice

Heinz Wrobel
Field Applications Engineer



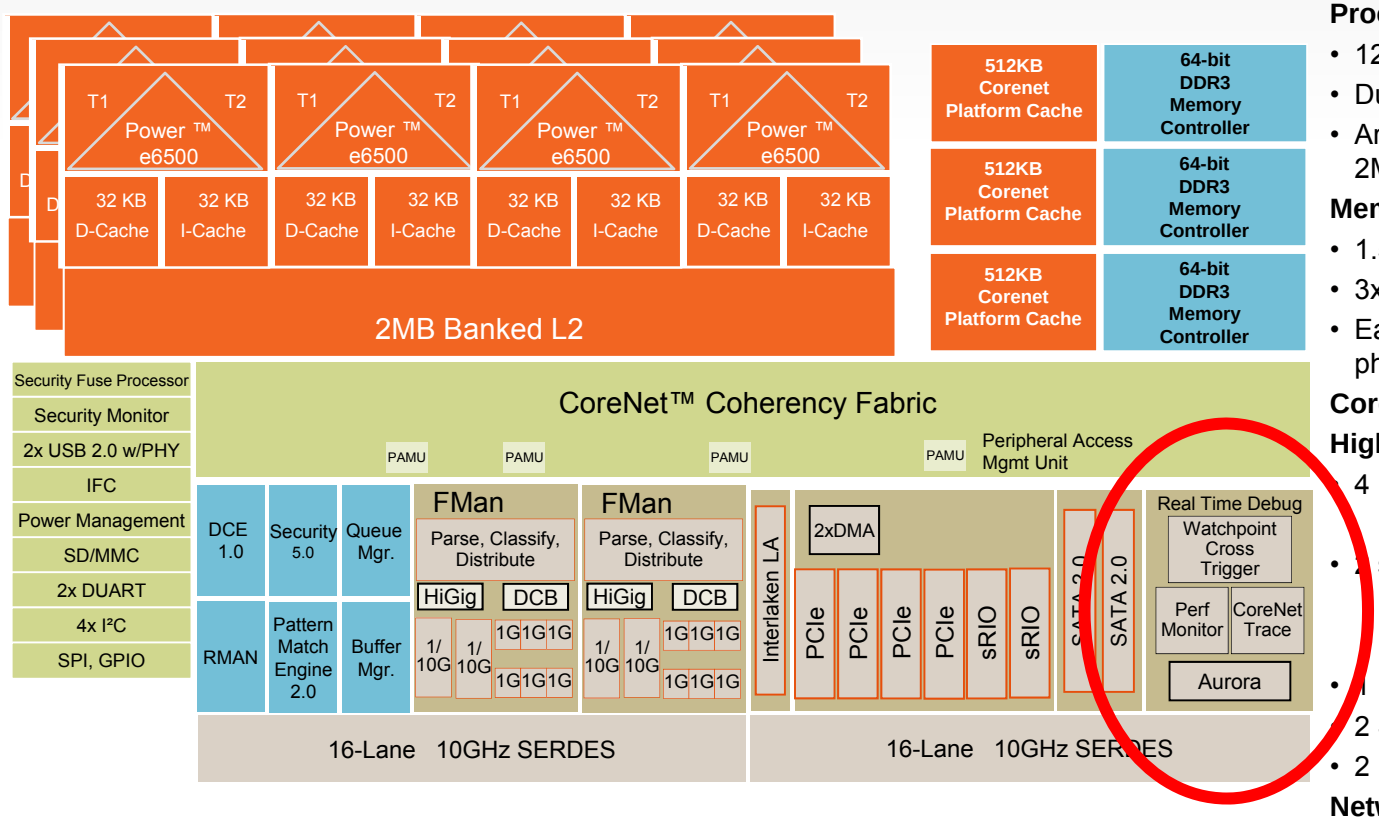
14-Nov-2013

Consider a real life customer project…

- An SMP Linux with RT patch as foundation
- Single applications having >20 threads
  - Control threads
  - Hard real time threads with hard deadlines
  - Soft real time threads
  - Shared objects loaded at runtime
- User space data path acceleration with DPAA
- Core affinity employed to allow realistic determinism
- Multiple other applications/services running concurrently

“Names and solution have been changed to protect the innocent”

Customer may use a T4240



Datapath Acceleration

- **SEC**- crypto acceleration 40Gbps
- **PME-** Reg-ex Pattern Matcher 10Gbps
- **DCE-** Data Compression Engine 20Gbps

Processor

- 12x e6500, 64b, up to 1.8GHz
- Dual threaded, with128b AltiVec
- Arranged as 3 clusters of 4 CPUs, with 2MB L2 per cluster; 256KB per thread

Memory SubSystem

- 1.5MB CoreNet Platform Cache w/ECC
- 3x DDR3 Controllers up to 2.1GHz
- Each with up to 1TB addressability (40 bit physical addressing)

CoreNet Switch Fabric

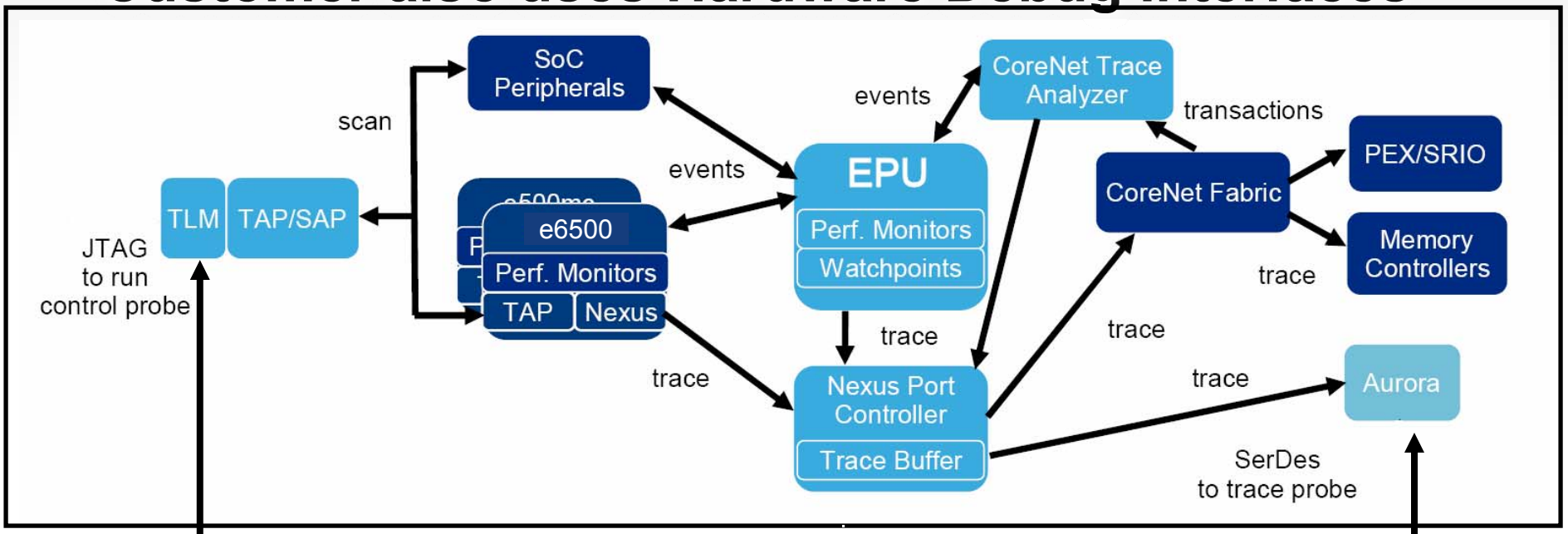
High Speed Serial IO

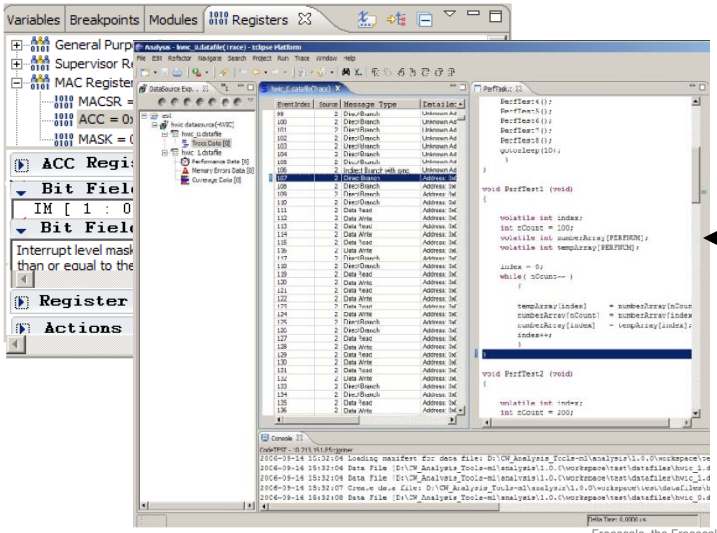
- 4 PCIe Controllers, with Gen3
  - SR-IOV support
- 2 sRIO Controllers
  - Type 9 and 11 messaging
  - Interworking to DPAA via Rman
- 1 Interlaken Look-Aside at up to 10GHz
- 2 SATA 2.0 3Gb/s
- 2 USB 2.0 with PHY

Network IO

- 2 Frame Managers, each with:
  - Up to 25Gbps parse/classify/distribute
  - 2x10GE, 6x1GE
  - HiGig, Data Center Bridging Support
  - SGMII, QSGMII, XAUI, XFI, KR

Customer also uses Hardware Debug Interfaces





Trace Overview

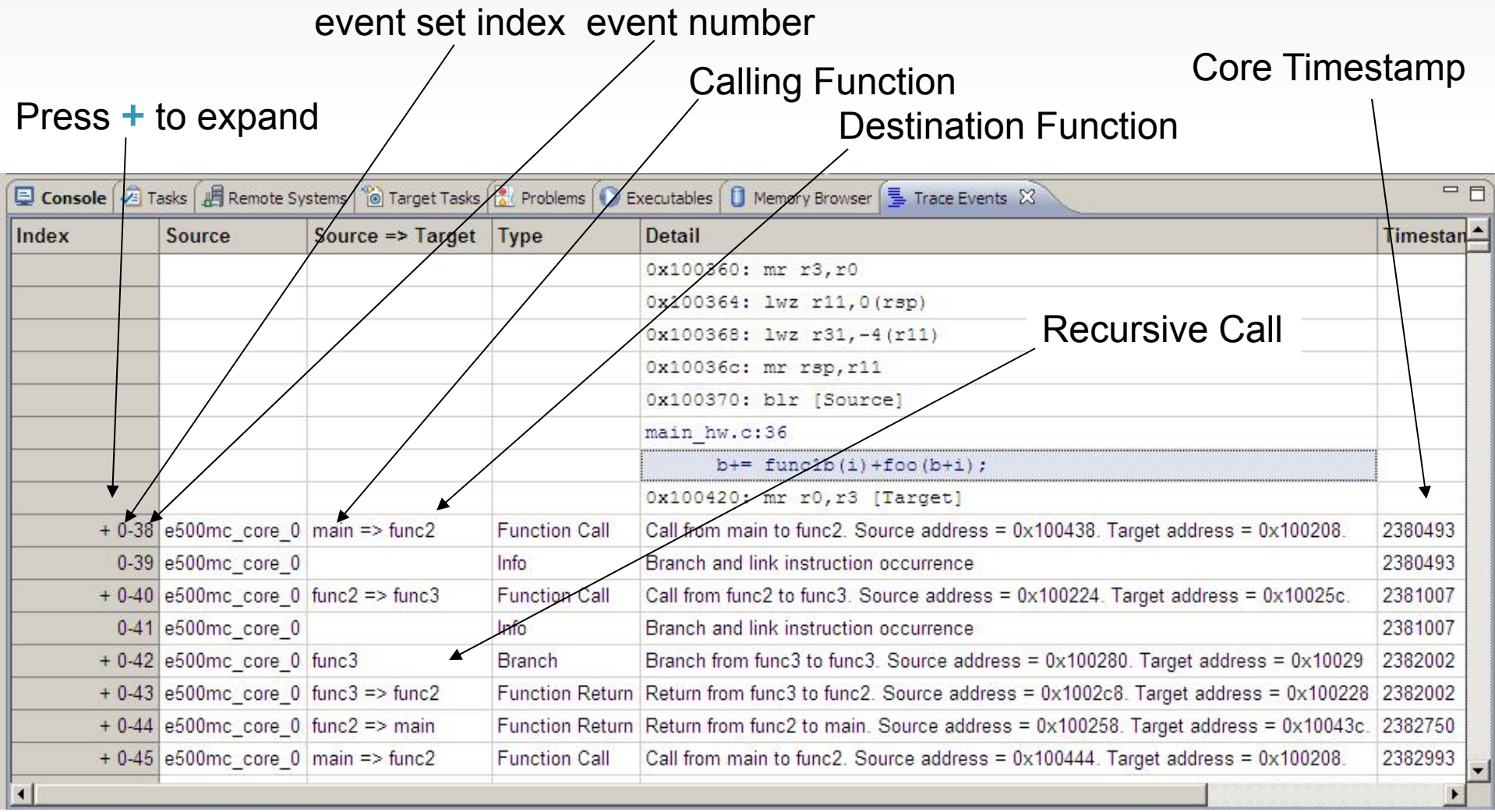
- The Trace functionality is split into three distinct parts:
  1. Trace Configuration
  2. Trace Collection/Decoding
  3. Trace View

- Trace Collection uses one of three methods:
  - **Nexus Trace Buffer**: 16K; Non-intrusive; small
  - **DDR buffer**: Limited by size of DDR; Intrusive; no DDR trace
  - **Aurora Trace**: Limited by probe memory; Non-intrusive; Aurora/Nexus socket on target board

I/O Physics: Trace Collection Limitations

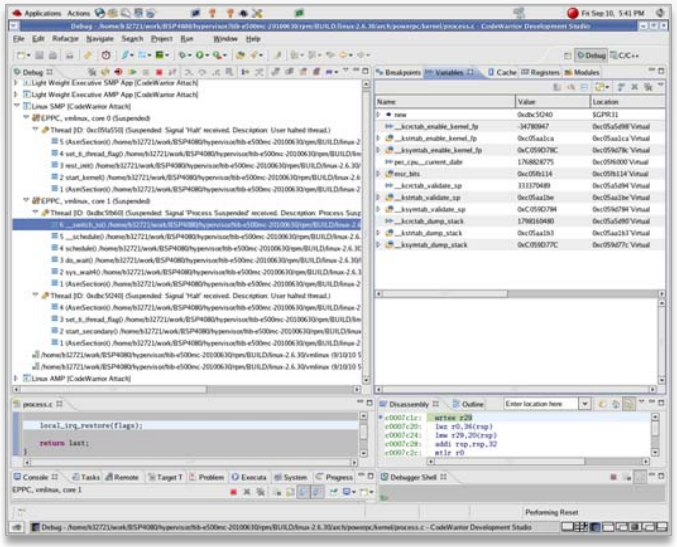
- Let's assume only a single core at 1.5GHz and start rounding …
- Assuming an average completion rate of 0.65 instructions per cycle: 1G instructions/second
- Each time-stamped Nexus program trace message: max 128bits
- Assuming a direct-to-indirect branch ratio of 9:1 – 128bits of trace for every 10 branches
- Assuming a linear-to-branch instruction ratio of 7:1 – 128bits of trace for every 80 instructions (1.6 bits per instruction)
- 1G instructions/sec x 1.6 bits of trace data = **1.6Gbps per core *(program trace only)***

Trace Events View

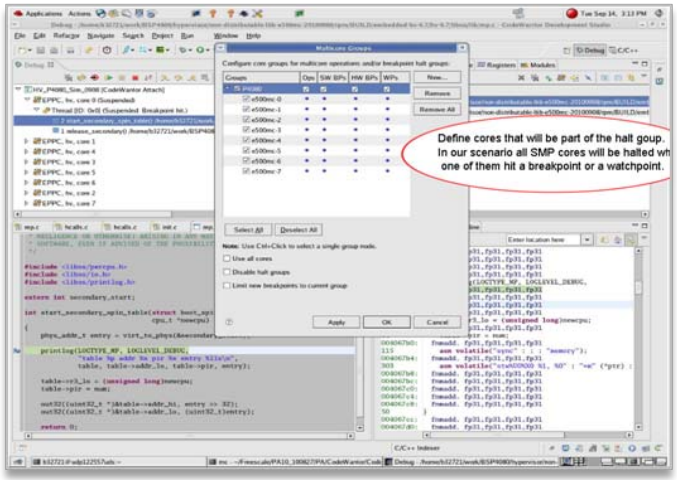


CodeWarrior

- **Multi [core/IC/type/thread/config]**
  - Multicore from the ground up
  - Multiple IC's
  - Multiple CPU types
  - Hardware threads
  - Mixed AMP/SMP
- **HW affinitive**
  - Breakpoints
  - Cross triggering
  - Tracing Analysis
  - Counters Analysis
- **Linux aware**
  - Kernel Debug via JTAG
  - App Debug via JTAG/UART/Eth
  - Tracks MMU state
- **Reference manual aware**



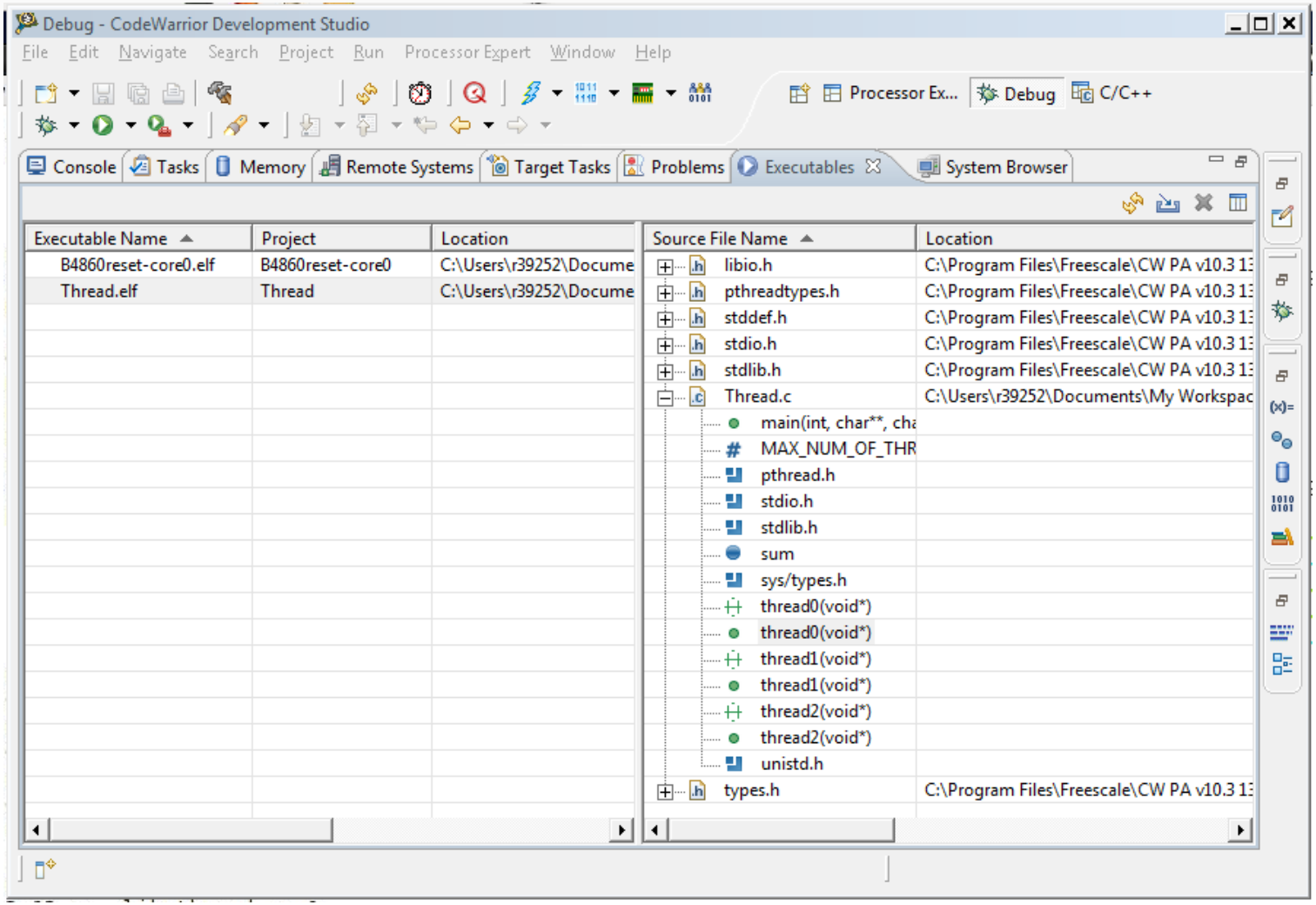
freescale CodeWarrior



http://www.freescale.com/CodeWarrior

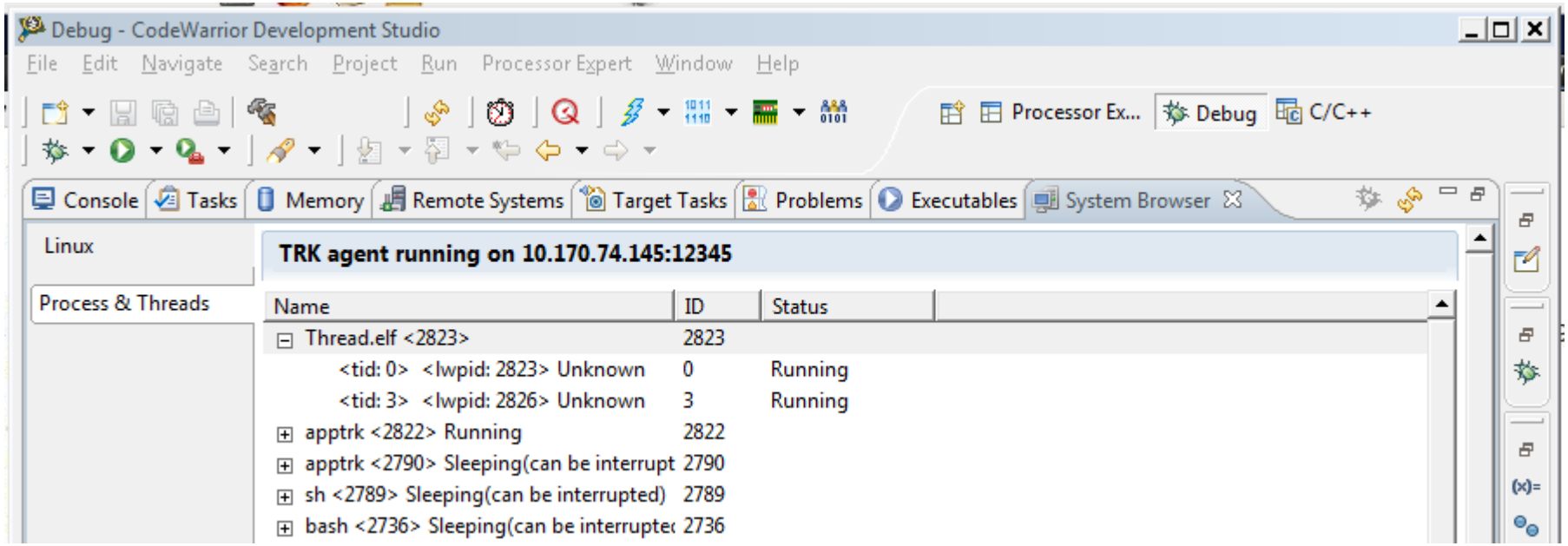
freescale™

The “Executables” view, offline analysis



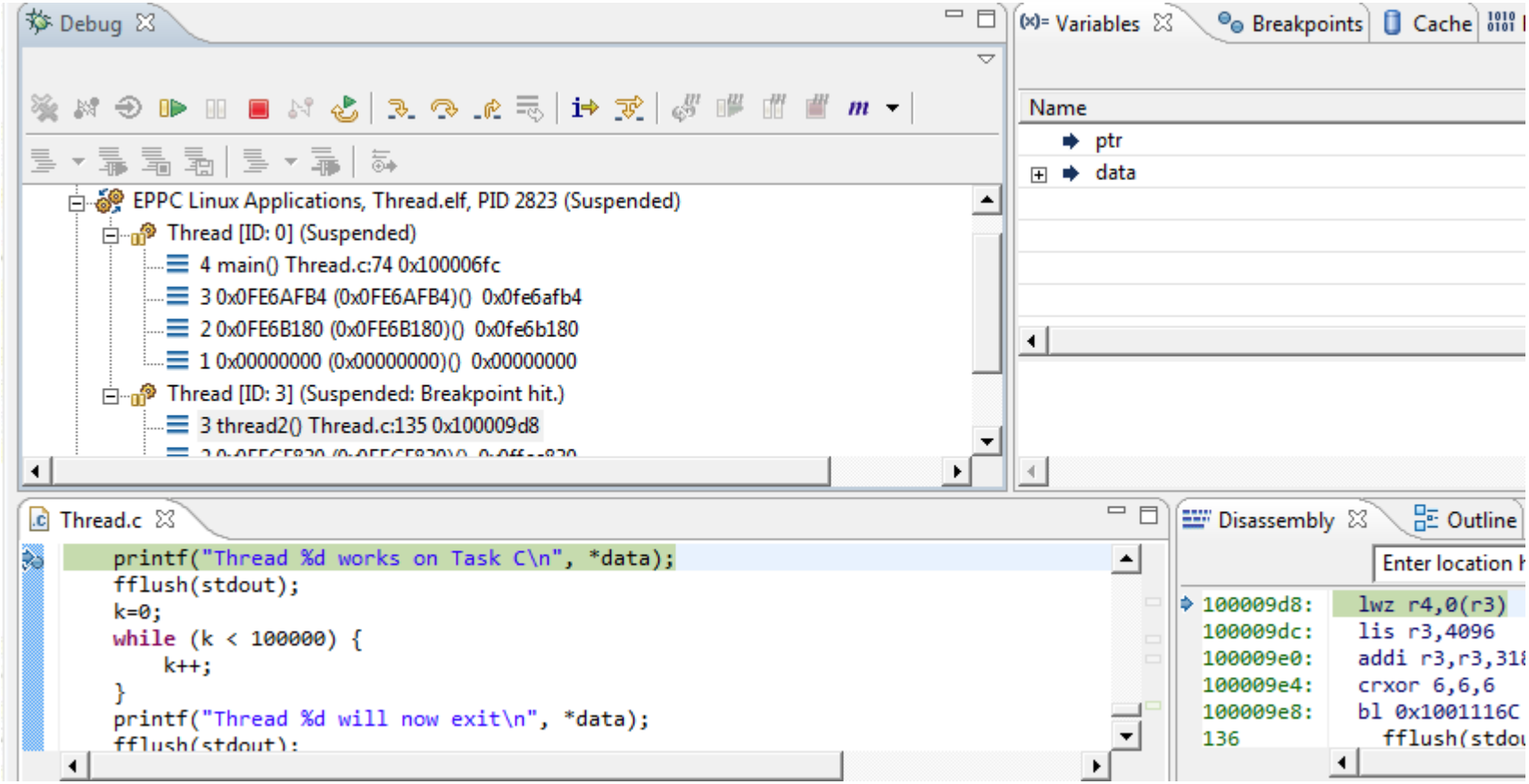
The “System Browser” view, online analysis

- You can map CodeWarrior threads displayed to target threads
- You can right click on a process to debug, if you have the .elf and, best case, sources!



Multi threaded debugging

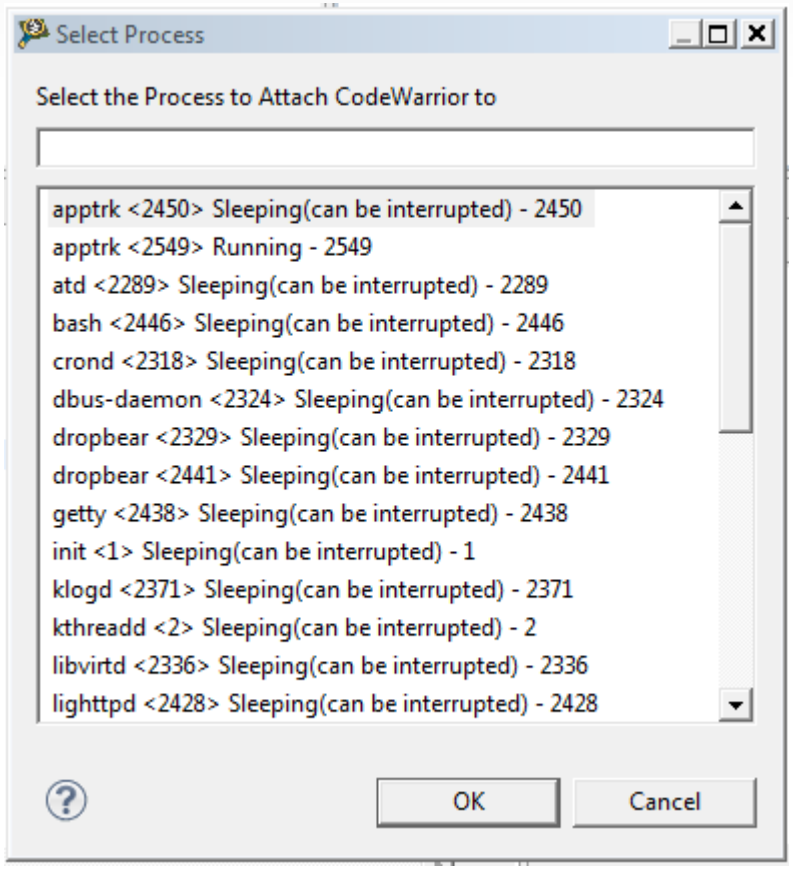
- CodeWarrior will grab threads created by the application



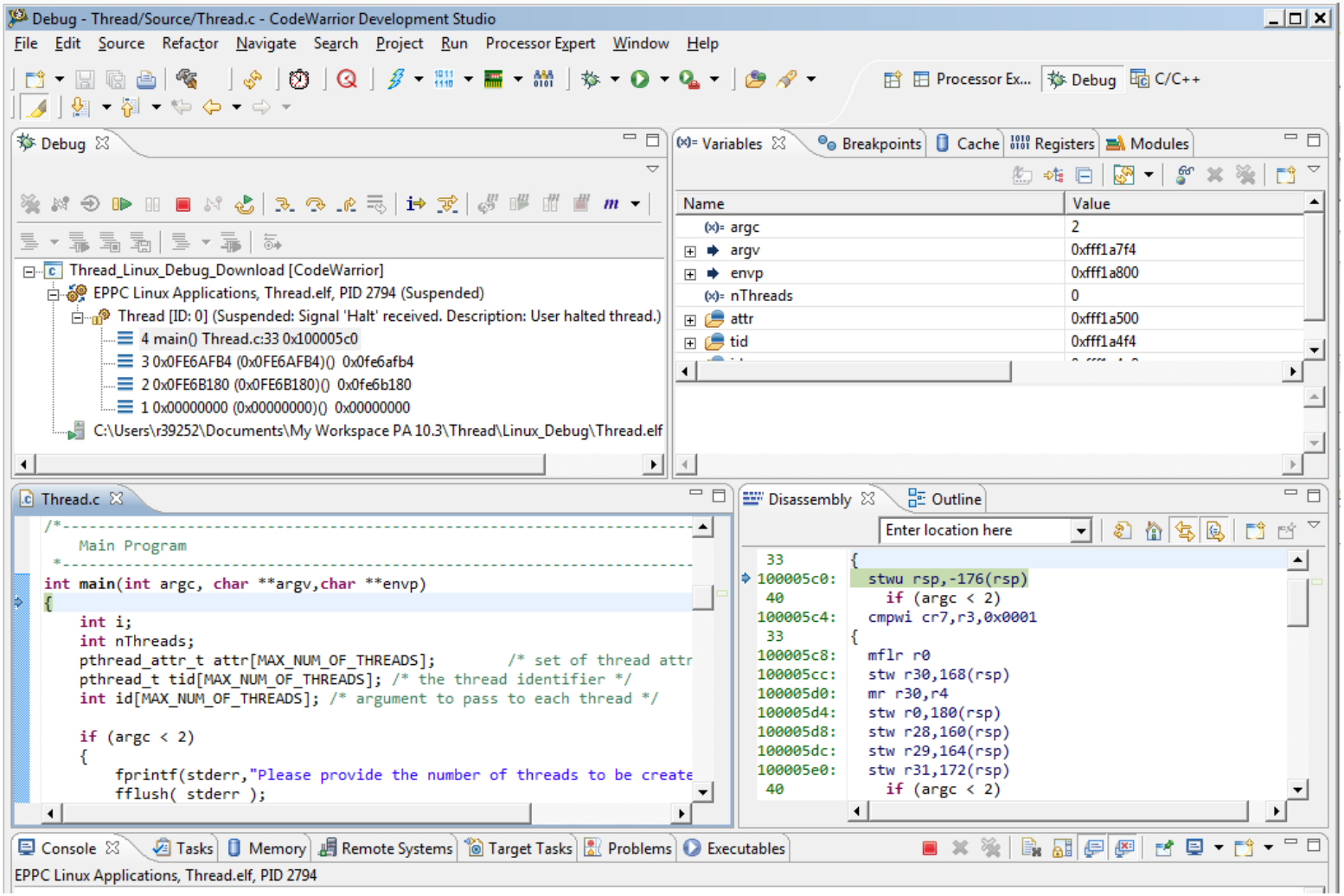
Attaching to a running process

- First prepare a debug setup that works well for a Download debug configuration
- Then change the debug configuration to session type “Attach”
- When starting debug, CodeWarrior will ask you

- Better pick the right one!



Start Debugging!?



What functionality is really used?

- CodeWarrior has tons of specific functionality for Multicore, but …
- Our brains tend to single task, so they have trouble …
  - Identifying context in highly multithreaded applications
  - Dealing with real AMP setups
  - Correlating events in substantially parallel systems
  - Dealing with complex tools if not used daily
- Tracing
  - I/O physics limit full system visibility
  - Very useful in very specific well defined cases
  - Touched by power users mostly
- Once the engineers use it, you get positive and useful feedback, but …
- **Moving software engineers away from [k]printf()/trial+error is hard.**

Let‘s discuss:

Efficient Multicore Debug is a usability + human problem to solve!

Technology is easy in comparison …


freescale™