



Debugging tool functionality for Multicore real-time applications under Linux

What key customers request
What works well in practice

Heinz Wrobel

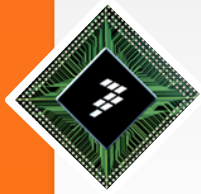
Field Applications Engineer



14-Nov-2013

Freescale, the Freescale logo, AllVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, mobileGT, PowerQUICC, Processor Expert, QorIQ, Qorivva, StarCore, Symphony and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QUICC Engine, Ready Play, SafeAssure, the SafeAssure logo, SMARTMOS, TurboLink, Vybrid and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2012 Freescale Semiconductor, Inc.



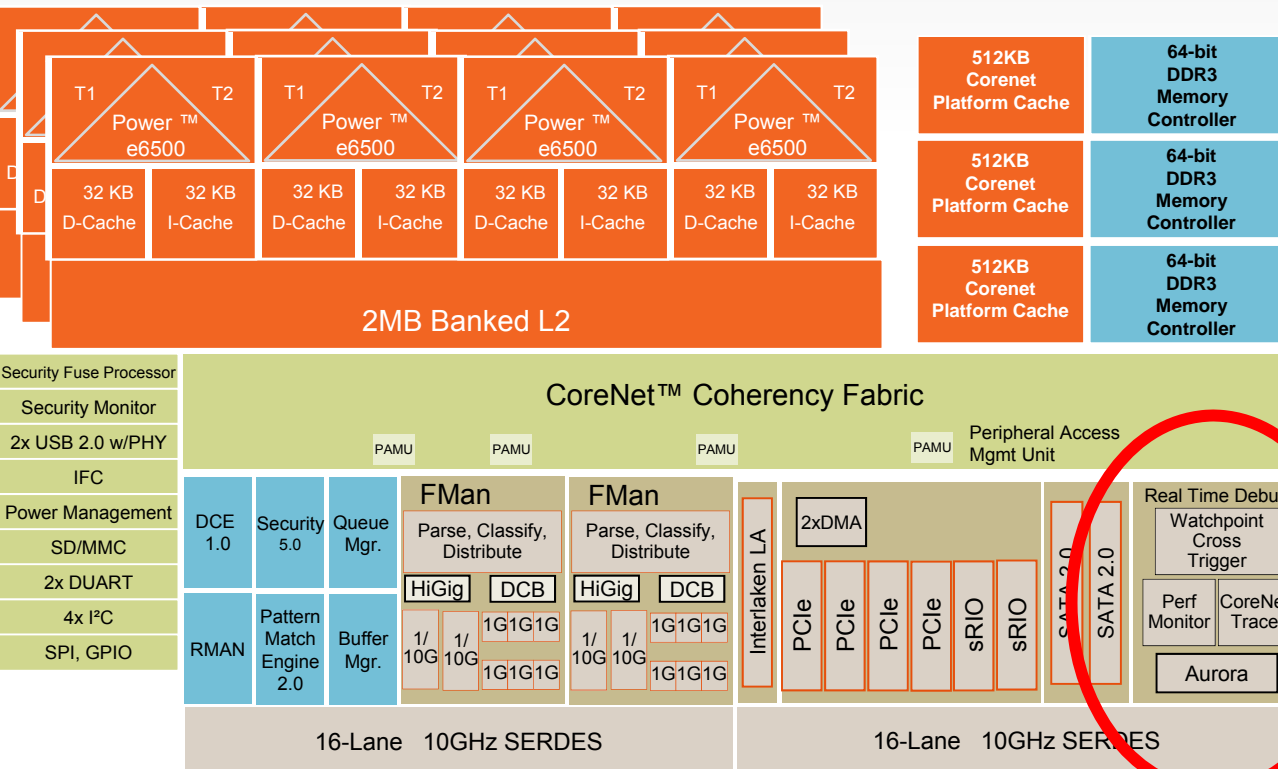


Consider a real life customer project...

- 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



Processor

- 12x e6500, 64b, up to 1.8GHz
- Dual threaded, with 128b 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

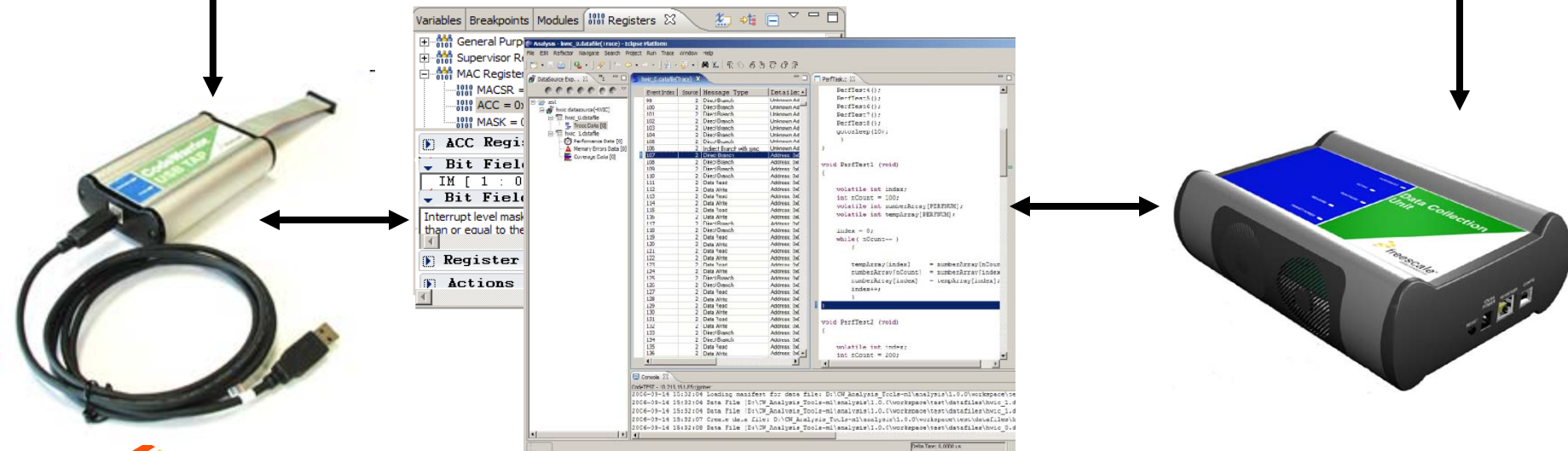
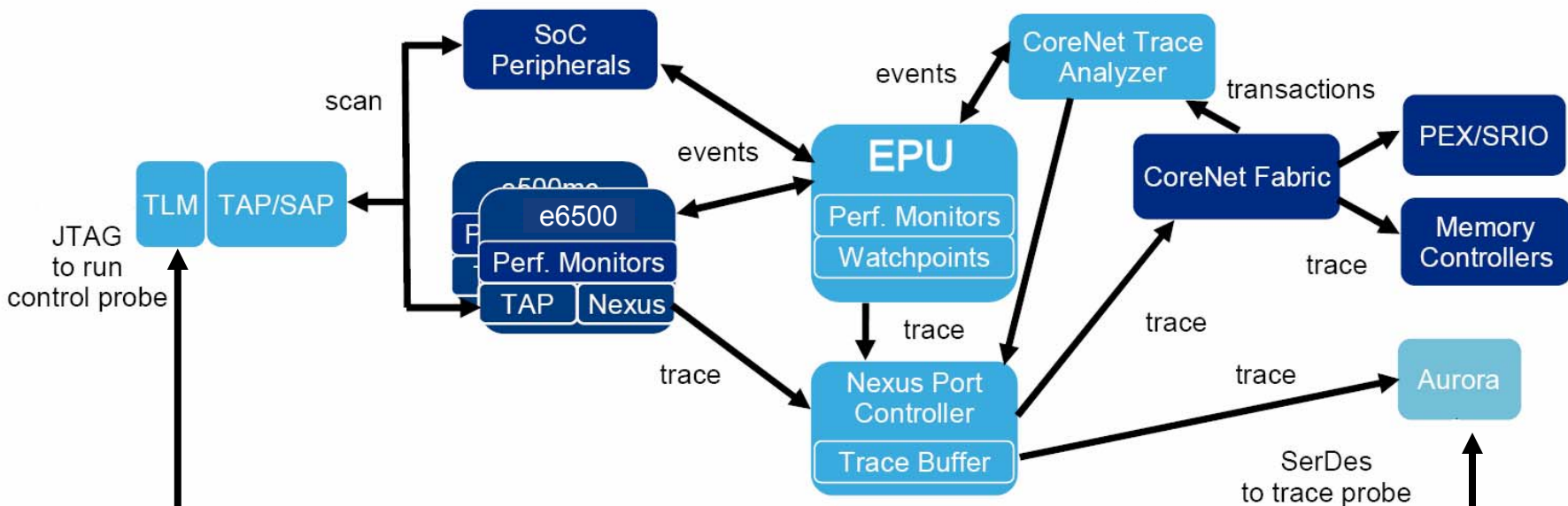
Network IO

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

Datapath Acceleration

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

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

event set index event number

Calling Function

Core Timestamp

Destination Function

Recursive Call

Press + to expand

Index	Source	Source => Target	Type	Detail	Timestamp
				0x100260: mr r3,r0 0x100364: lwz r11,0(rsp) 0x100368: lwz r31,-4(r11) 0x10036c: mr rsp,r11 0x100370: blr [Source] main_hw.c:36 b+= func1b(i)+foo(b+i); 0x100420: mr r0,r3 [Target]	
+ 0-38	e500mc_core_0	main => func2	Function Call	Call from main to func2. Source address = 0x100438. Target address = 0x100208.	2380493
0-39	e500mc_core_0		Info	Branch and link instruction occurrence	2380493
+ 0-40	e500mc_core_0	func2 => func3	Function Call	Call from func2 to func3. Source address = 0x100224. Target address = 0x10025c.	2381007
0-41	e500mc_core_0		Info	Branch and link instruction occurrence	2381007
+ 0-42	e500mc_core_0	func3	Branch	Branch from func3 to func3. Source address = 0x100280. Target address = 0x10029	2382002
+ 0-43	e500mc_core_0	func3 => func2	Function Return	Return from func3 to func2. Source address = 0x1002c8. Target address = 0x100228	2382002
+ 0-44	e500mc_core_0	func2 => main	Function Return	Return from func2 to main. Source address = 0x100258. Target address = 0x10043c.	2382750
+ 0-45	e500mc_core_0	main => func2	Function Call	Call from main to func2. Source address = 0x100444. Target address = 0x100208.	2382993

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

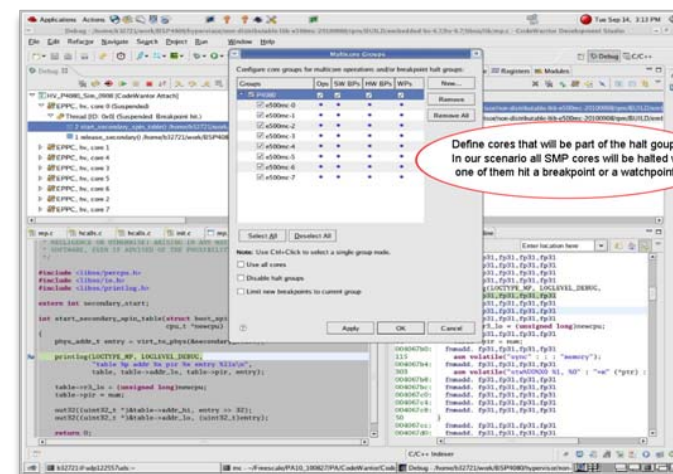
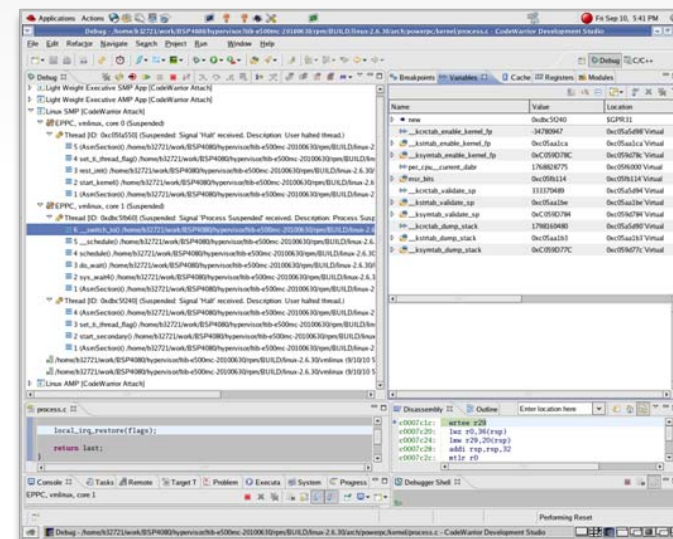
- Breakpoints
- Cross triggering
- Tracing Analysis
- Counters Analysis



• Linux aware

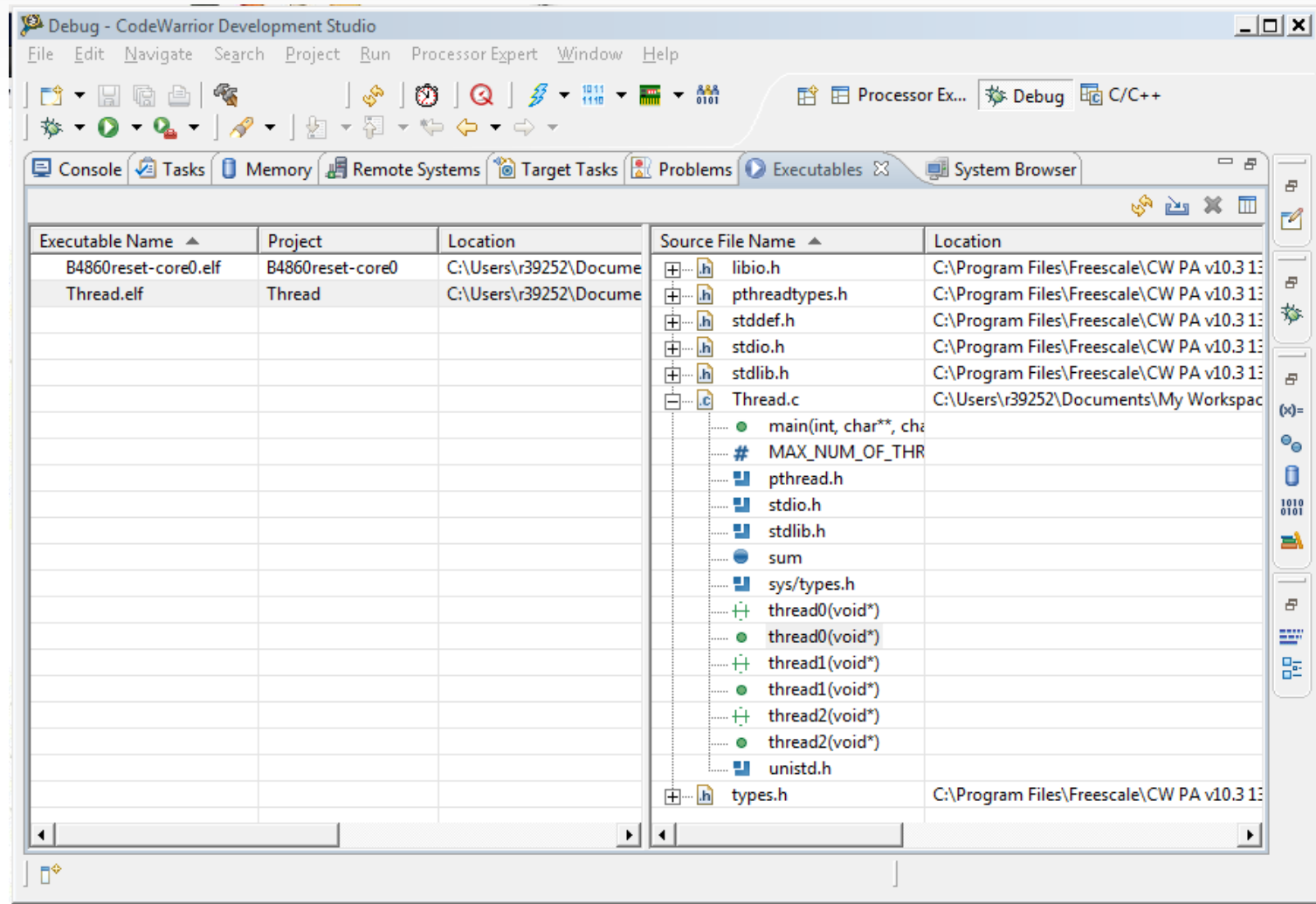
- Kernel Debug via JTAG
- App Debug via JTAG/UART/Eth
- Tracks MMU state

• Reference manual aware



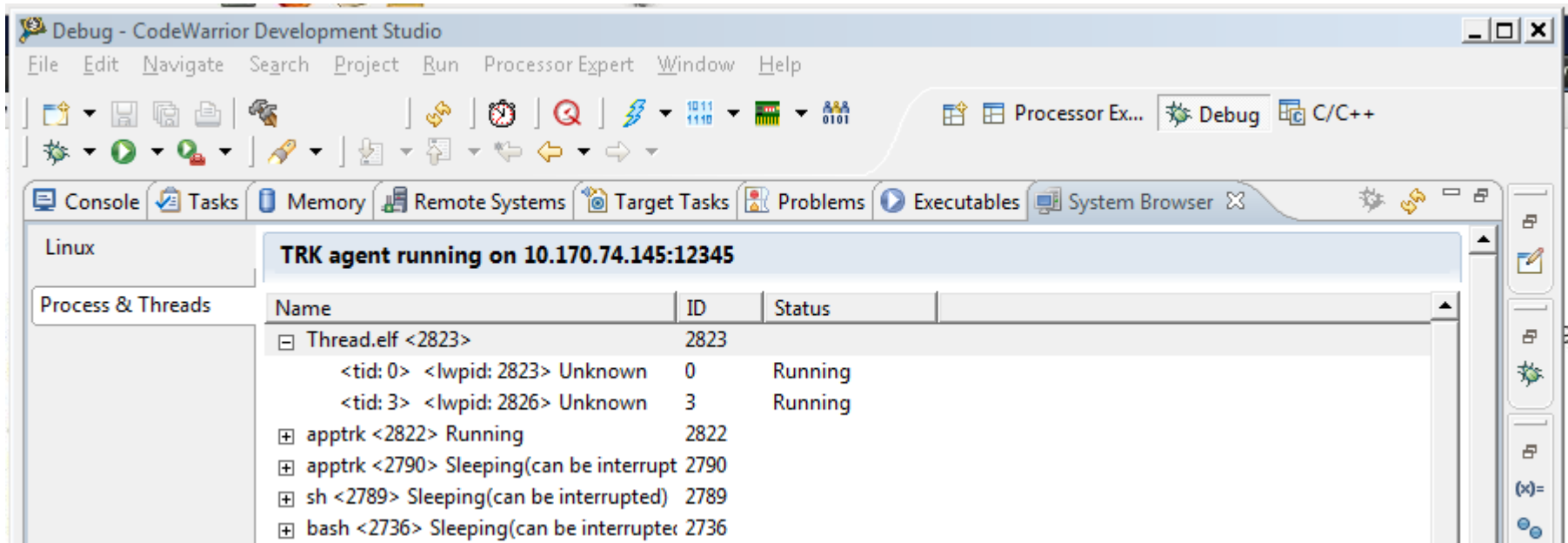
<http://www.freescale.com/CodeWarrior>

The “Executables” view, offline analysis



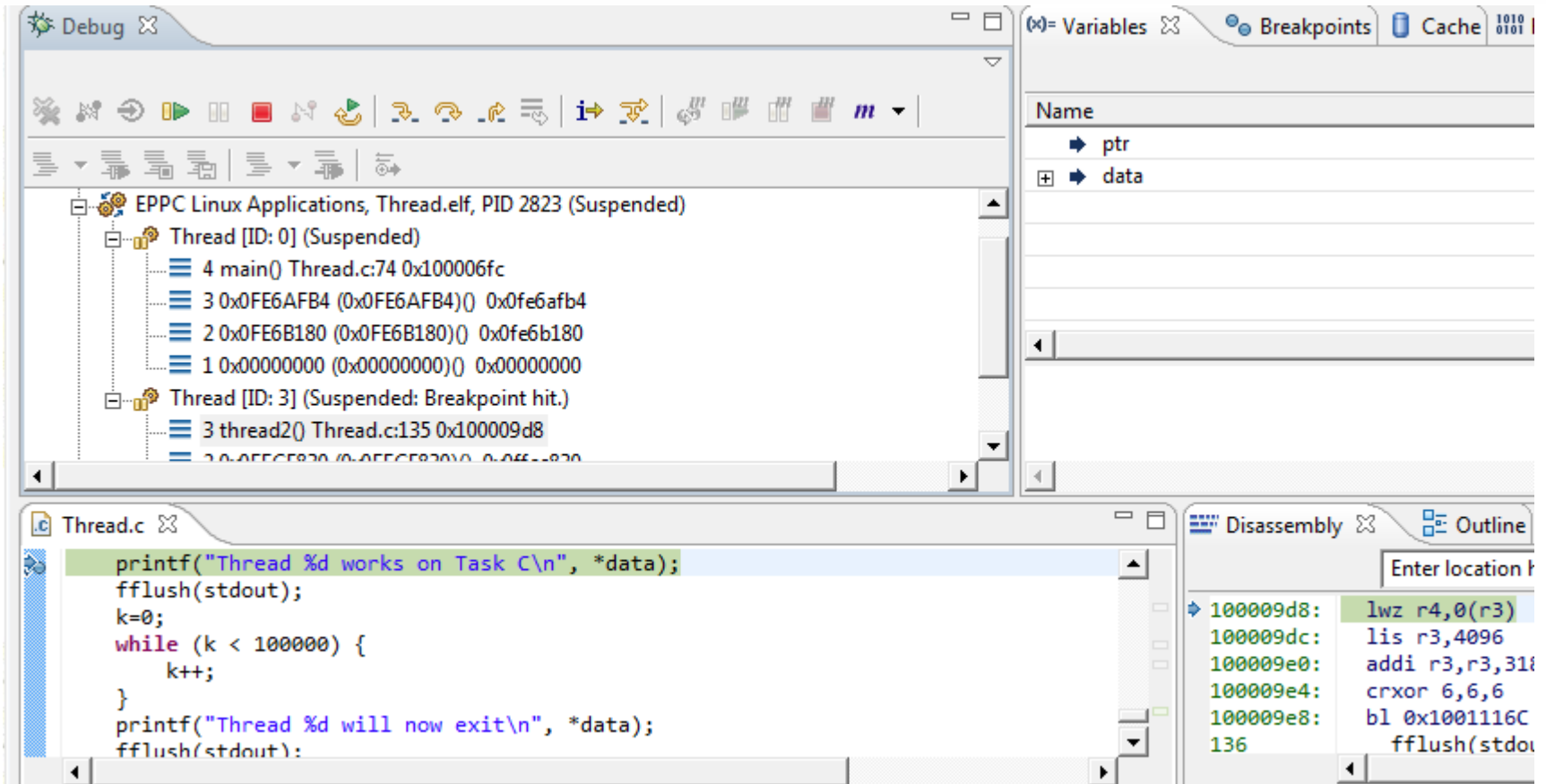
The “System Browser” view, online analysis

- 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!



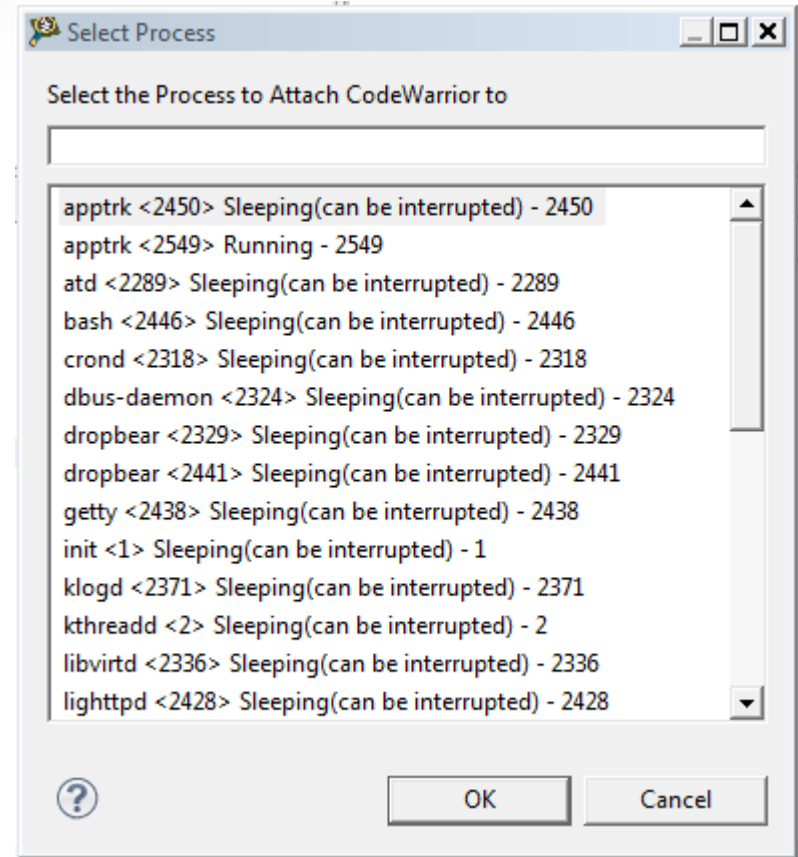
Multi threaded debugging

- 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!?

The screenshot displays the CodeWarrior Development Studio interface during a debug session. The main window shows the source code of `Thread.c` with the `main` function highlighted. The `main` function is defined as follows:

```
int main(int argc, char **argv, char **envp)
{
    int i;
    int nThreads;
    pthread_attr_t attr[MAX_NUM_OF_THREADS]; /* set of thread attr
    pthread_t tid[MAX_NUM_OF_THREADS]; /* the thread identifier */
    int id[MAX_NUM_OF_THREADS]; /* argument to pass to each thread */

    if (argc < 2)
    {
        fprintf(stderr, "Please provide the number of threads to be create
        fflush( stderr );
    }
}
```

The `Debug` window on the left shows the thread list with the following details:

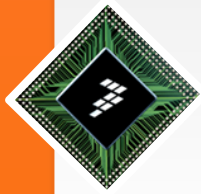
- Thread [ID: 0] (Suspended: Signal 'Halt' received. Description: User halted thread.)
- 4 main() Thread.c:33 0x100005c0
- 3 0x0FE6AFB4 (0x0FE6AFB4)() 0x0fe6afb4
- 2 0x0FE6B180 (0x0FE6B180)() 0x0fe6b180
- 1 0x00000000 (0x00000000)() 0x00000000

The `Variables` window on the right shows the following variables and their values:

Name	Value
argc	2
argv	0xffff1a7f4
envp	0xffff1a800
nThreads	0
attr	0xffff1a500
tid	0xffff1a4f4

The `Disassembly` window on the right shows the assembly code for the `main` function, starting at address 100005c0:

```
100005c0: stwu rsp, -176(rsp)
100005c4: if (argc < 2)
100005c4: cmpwi cr7, r3, 0x0001
100005c8: mflr r0
100005cc: stw r30, 168(rsp)
100005d0: mr r30, r4
100005d4: stw r0, 180(rsp)
100005d8: stw r28, 160(rsp)
100005dc: stw r29, 164(rsp)
100005e0: stw r31, 172(rsp)
100005e0: if (argc < 2)
```

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 ...

