Overview
IAR Embedded Workbench for ARM

Infineon XMC 45xx TechDay

Munich, June 25 2014

Milan, July 1 2014

Martin Gisbert
FAE IAR Systems

IAR SYSTEMS

Agenda

- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

Agenda

- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

What is IAR Embedded Workbench?

IAR Embedded Workbench IDE

IDE Tools

Editor

Project manager

Library builder

Librarian

Build Tools

IAR C/C++ Compiler

Assembler

Linker

IAR C-SPY Debugger

Simulator driver

Hardware system drivers

Power debugging

RTOS plug-ins

C-SPY Macros

Plug-ins

IDE

eclipse READY

Editors

Source code control systems

SUBVERSION

Project Connections

Configuration tools

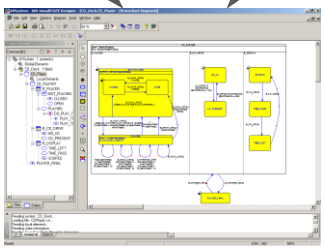
Infineon DAVE™

TÜV SÜD

Validation Reports

C-RUN

Runtime Analysis



visualSTATE

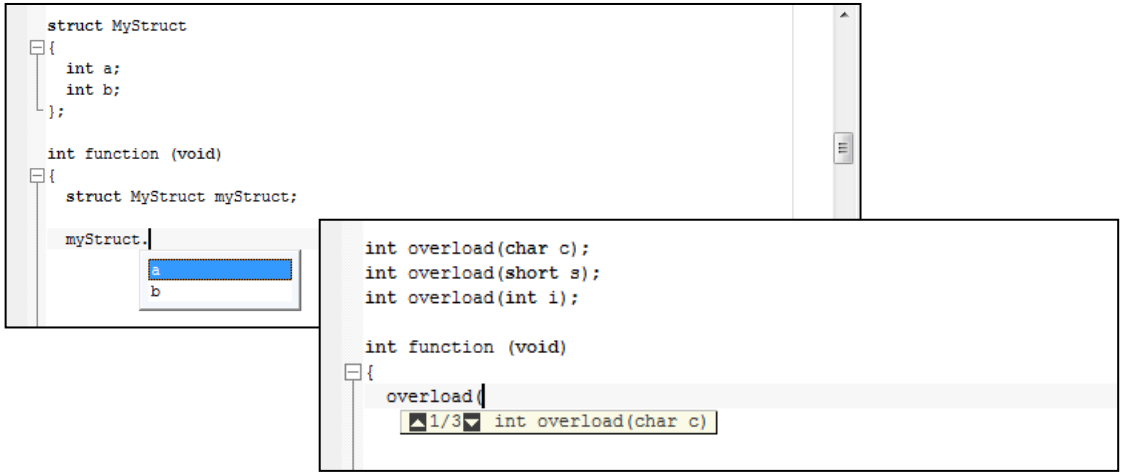
Strong ecosystem

Agenda

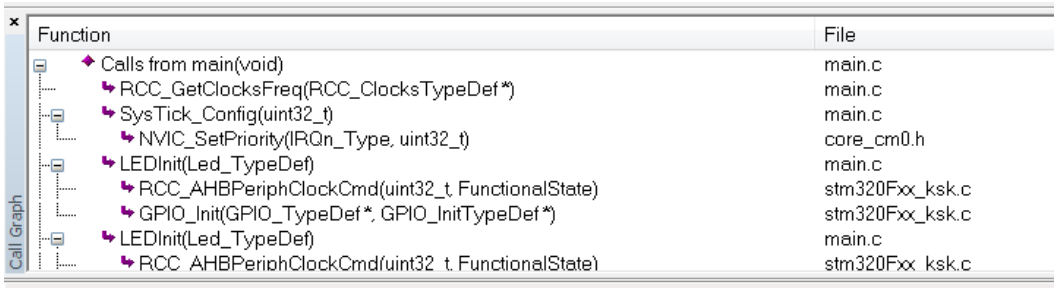
- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

IAR IDE Editor

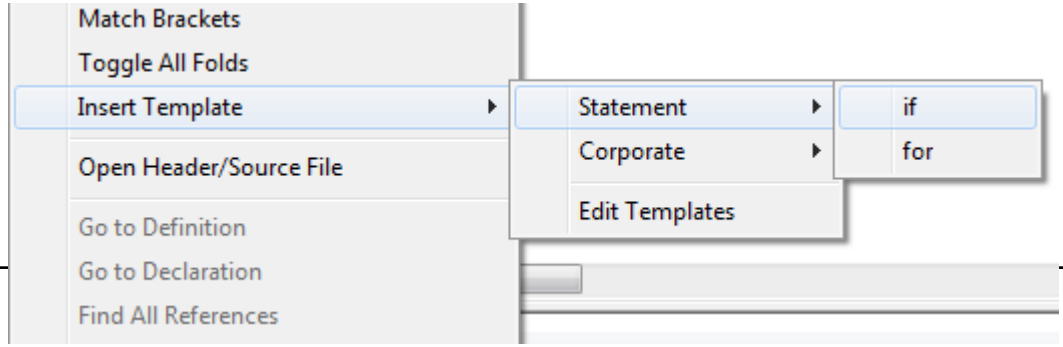
- Improved Editor with
  - code and comment folding
  - Word and Code completion
  - Parameter Hint



- Source Browser
  - Reliable „Go to Definition“
  - Find All Calls to/from <function> (Call Graph)



- Code templates

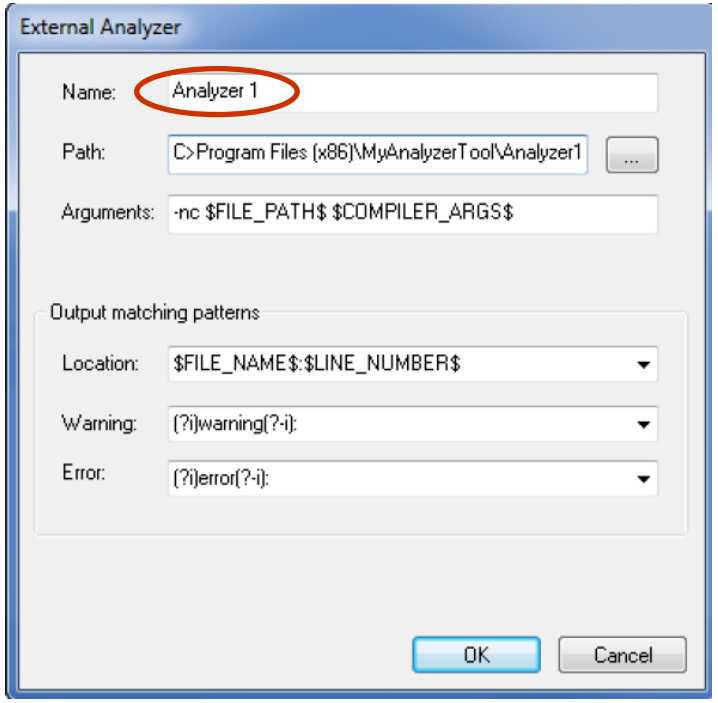


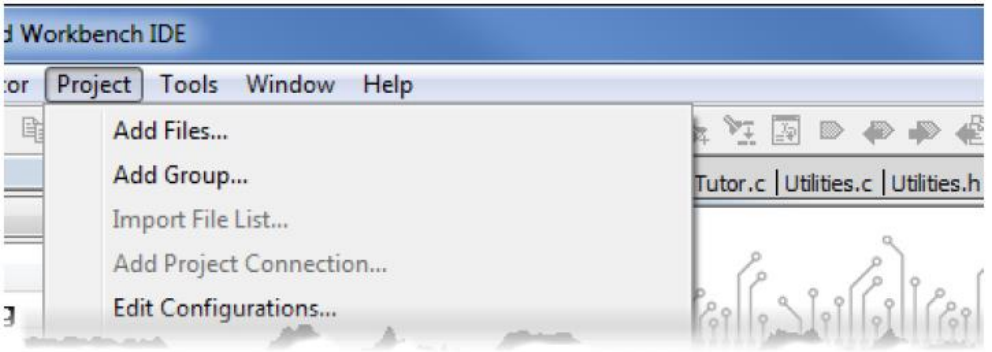
Project Manager, Project Options

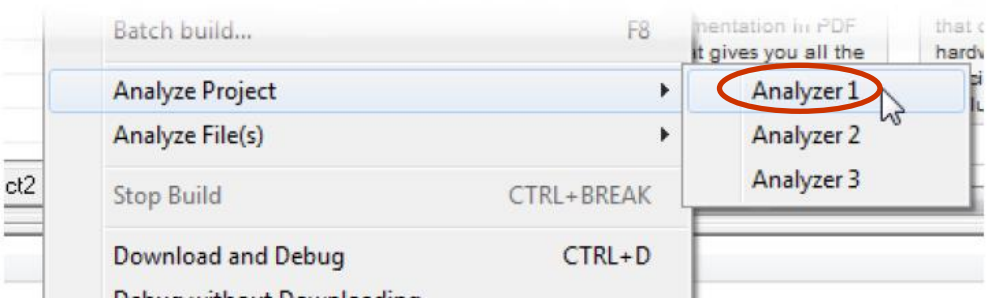
- Pre-/Post-build actions
  Invokes any utility before or after the build process
- Output converter (elf/dwarf -> .bin, .hex, .s19, …)
- Project configurations (e.g. debug, release, Flash, RAM, …)
  individual options and included files per configuration
- Batch build for multiple projects/configurations within a workspace
- Shortcut to external tools

Adding external Analyzer tools

Tools -> Options -> Project -> External Analyzers







Eclipse Plugin

- Eclipse Plugin for EWARM
  Includes code generation **and** C-SPY debugger
- Supports Simulator, I-jet, JTAGjet, J-Link and J-Trace probes
- Project converter to import EWARM projects into Eclipse and update imported projects
- More Eclipse plugins under www.iar.com/eclipse

IAR Embedded Workbench for Eclipse

http://iar.com/eclipse

Installation

To install IAR Embedded Workbench for Eclipse, choose **Help>Install New Software** and use the update site bel

Requirements

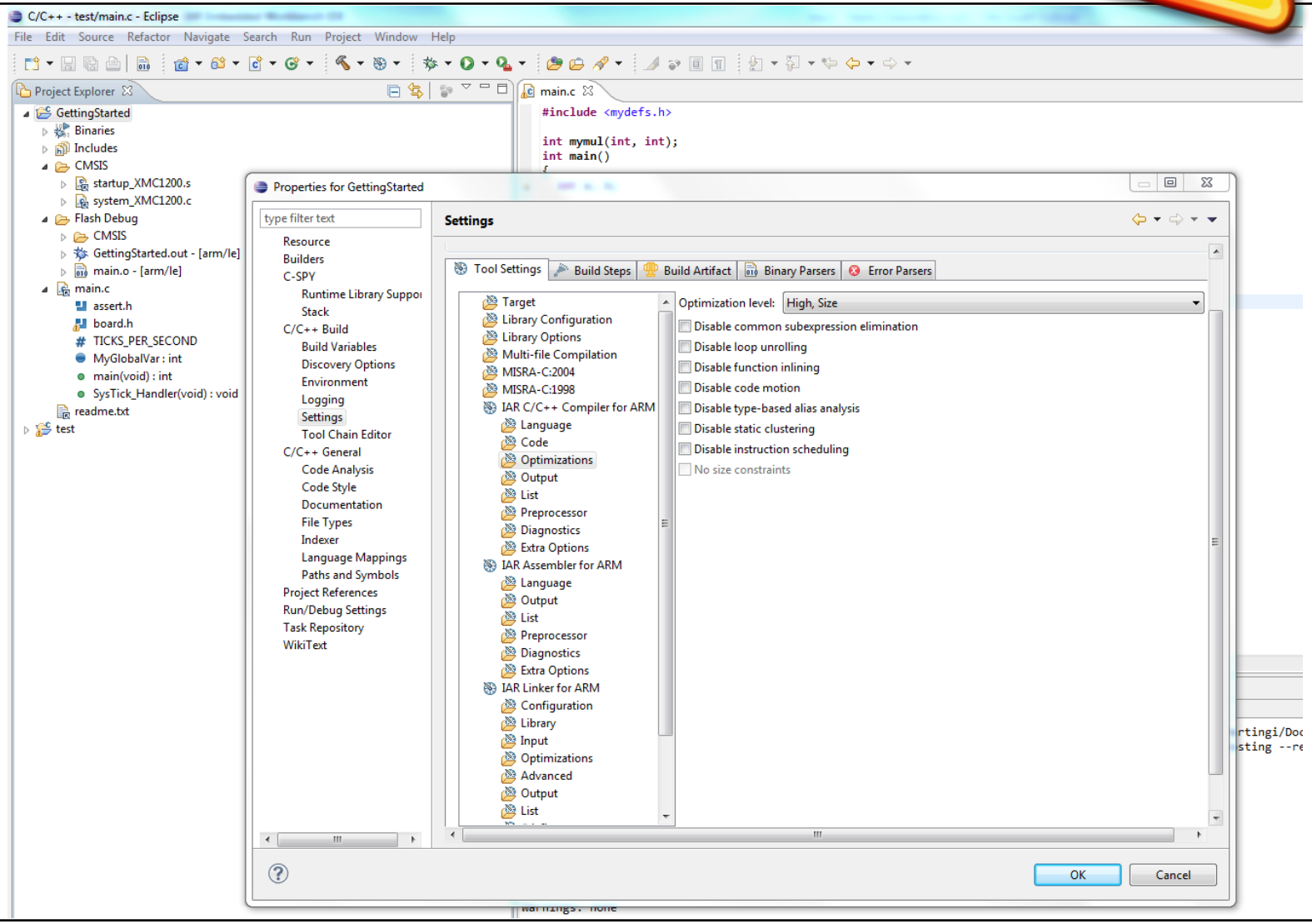
- **Eclipse**. Download Eclipse.
- **IAR Embedded Workbench** (the IAR Eclipse plugins do not include the compiler or debugger itself).
- **Java Runtime Environment**, version 6 or later. Download Java.

ARM

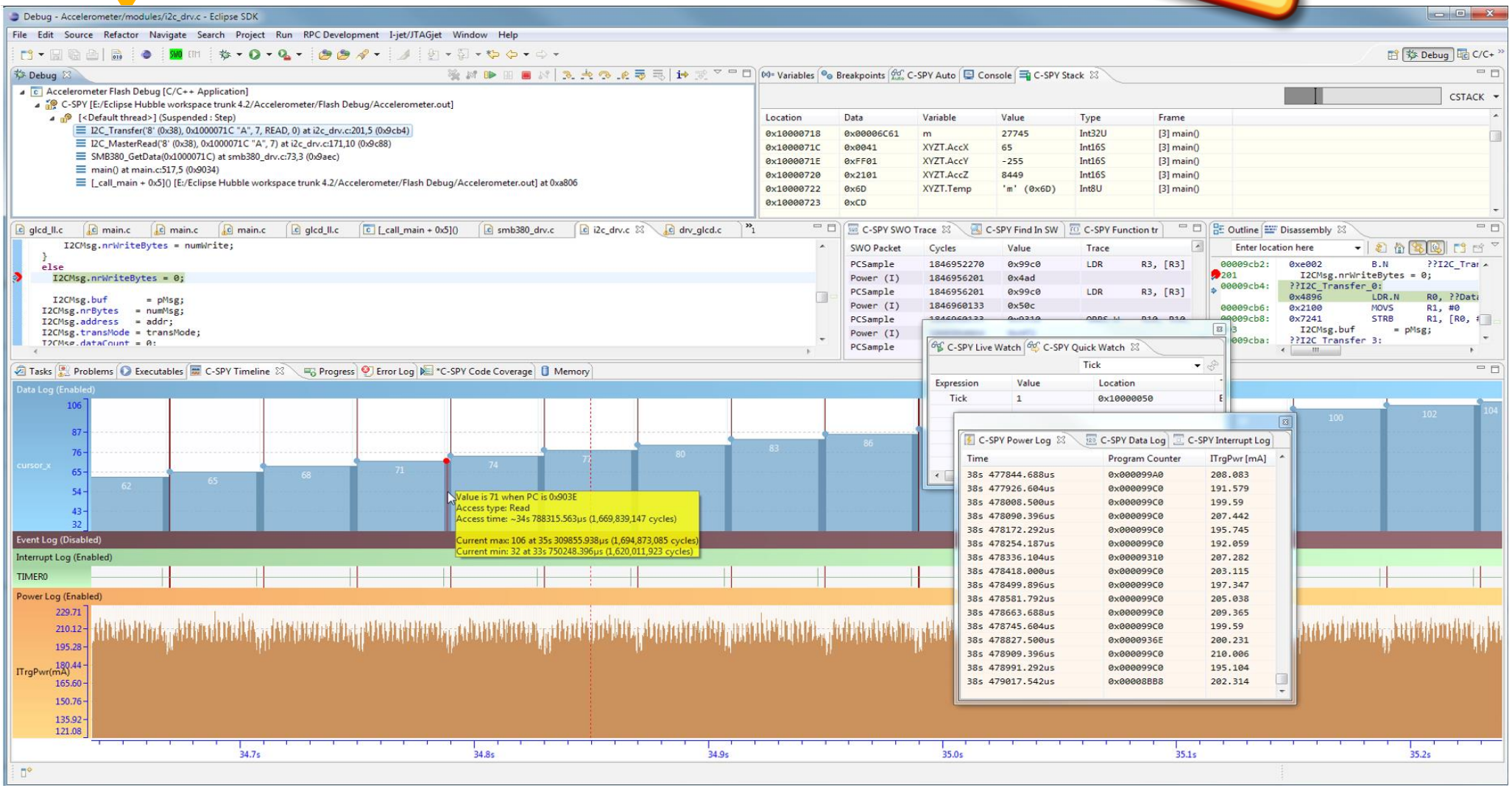
Includes build **and** debug support.

EW version	Supported Eclipse platforms Indigo and later
6.50.x/6.60.x/6.70.x	update-site | zip | release notes (build 12, qualifier 201403071414)
6.40.x	update-site | zip | release notes (build 58, qualifier v20131125-134024)
6.30.x	update-site | zip | release notes (build 56, qualifier v20130930-152250)
6.21.x	update-site | zip | release notes (build 36, qualifier 201308202349)
6.10.x	update-site | zip | release notes (build 167, qualifier 201308230935)

Eclipse Plugin (Code View)



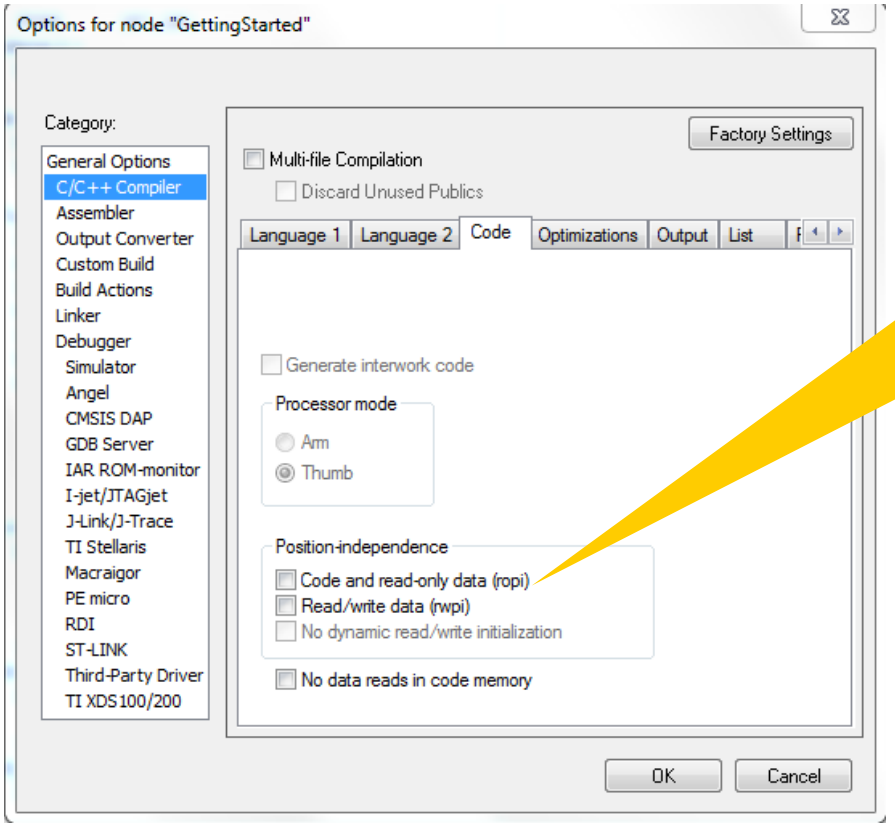
Eclipse Plugin (Debugging View)



Agenda

- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

Powerful C/C++ Compiler (Cont'd)

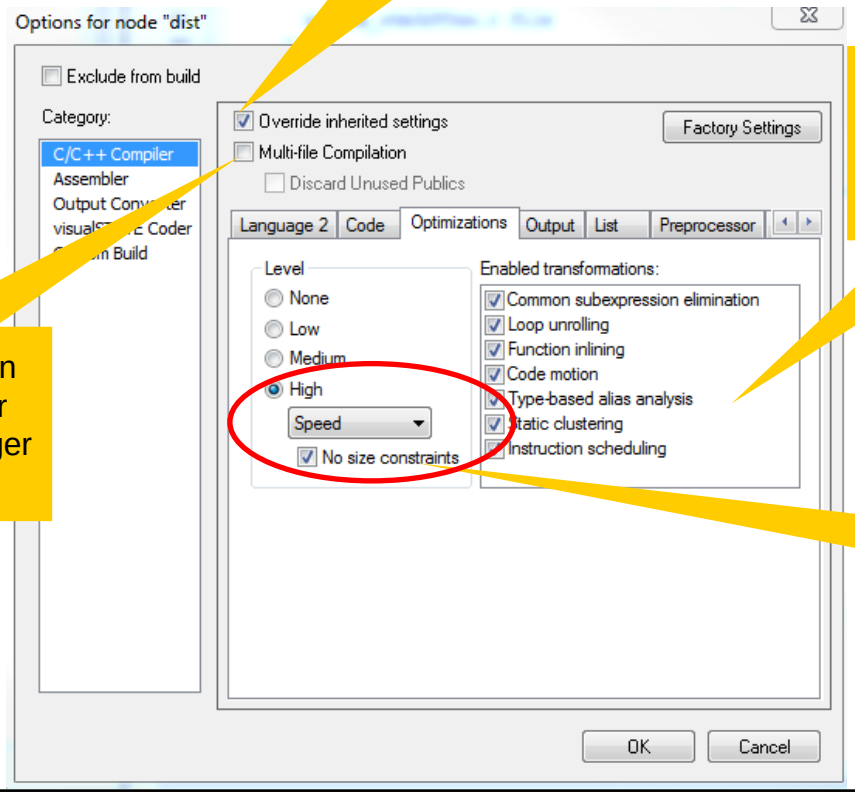


Support for position independent code and data

ropi generates code that uses PC-relative references to address code and read-only data

Powerful C/C++ Compiler

Balance between size and speed can be achieved by setting different optimizations for different parts of the code



Multi-file compilation allows the optimizer to operate on a larger set of code

Major functions of the optimizer can be controlled individually

Speed option “no size constraints”

Well-tested

- Commercial test suites
  - Plum-Hall
  - Perennial
  - Dinkumware library test
- In-house developed test suite >500,000 lines of C/C++ test code run multiple times
  - Processor modes
  - Memory models
  - Optimization levels

Language standards

- ISO/IEC 9899:1990 (C94/C90/C89/ANSI C)
- ISO/IEC 9899:1999 (C99/Standard C)
- ISO/IEC 1488:2003 (Standard C++)
- Embedded C++ and Extended Embedded C++ dialects are also supported

Coremark XMC4500???

Agenda

- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

ILINK Linker - .icf file

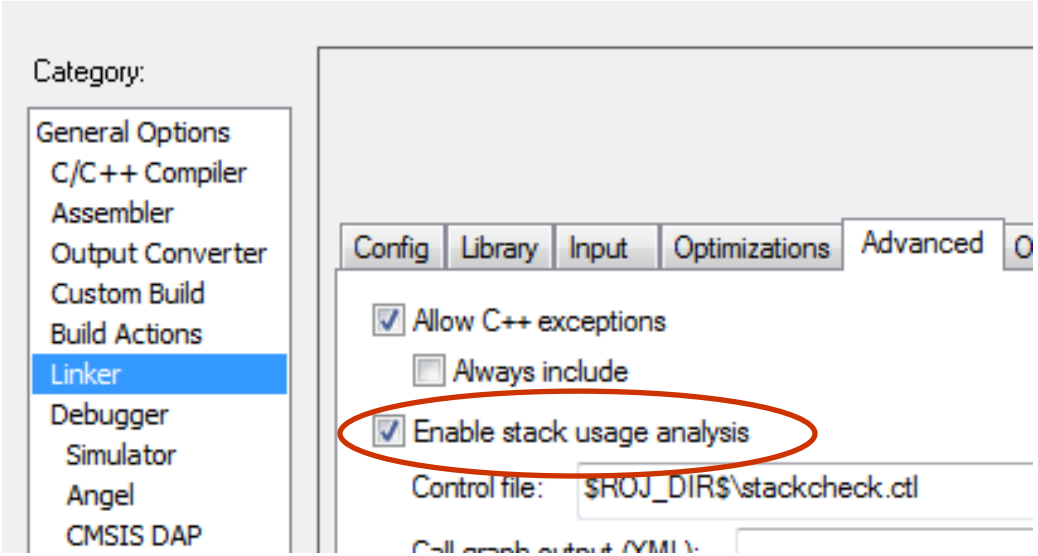
Placement controlled via text file

Ilink supports several types of section selectors:

- section attribute (ro, ro code, rw, rw data, zi, etc.)
  - `place ... { ro code };`
- section name
  - `place ... { section .text };`
- module name(file name)
  - `place ... { ro code object vector.o (vectRoutine.a)};`

Linker - Stack usage analysis

Under the right circumstances, the ILINK linker can calculate the maximum stack usage for each call graph root.



Linker - Stack usage analysis

Missing information, e.g. in case of recursion or indirect function calls, can be provided through pragmas or a *stack usage control file*:

```
function Reset_Handler:0, calls SystemInit, calls __iar_program_start;

call graph root [main_root]: Reset_Handler;
call graph root [interrupt]: NMI_Handler, SysTick_Handler, [..]

possible calls USBH_Init: USBH_USR_Init;

exclude __aeabi_i2d, __aeabi_ui2d;
max recursion depth GLCD_SendCmd: 2;
```

stackcheck.ctl

Linker - Stack usage analysis

- Stack usage chapter in the linker map file, listing of maximum stack depth for each call graph root.
- Alternative generation of a .xml call graph file
- Use linker extra option `--log call_graph` for better understanding

```
****************
*** STACK USAGE
***

  Call Graph Root Category  Max Use  Total Use
  ------------------------  -------  ---------
  interrupt                       8          8
  main_root                     164        164

main_root
  "__iar_program_start": 0x08009dcd

  Maximum call chain                                  164 bytes

    "__iar_program_start"                               0
    "__cmain"                                           0
    "main"                                              8
    "DrawPicture"                                      24
    "ResetPicture"                                     16
    "GLCD_SendCmd"                                     32
      + 1 cycles in nest 0                             32
    "GLCD_SendCmd"                                     32
    "GLCD_SPI_ReceiveBlock"                            16
    "GLCD_SPI_TranserByte"                              4
...
```

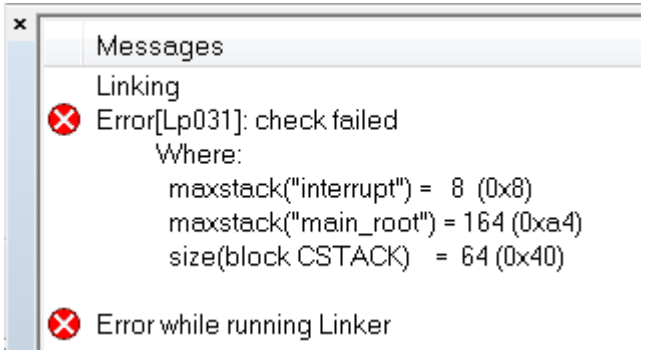
.map

Linker - Stack usage analysis

- Optional „check“ compares the stack analysis result with the actual size of the stack
- If the result of the analysis exceeds the size of the CSTACK block, the linker generates an error:

```
[…]
check that
    maxstack("main_root")
    + maxstack("interrupt")    // use totalstack()
                               // if preemptive
    + 000                      // optional margin
    <= size (block CSTACK);
```

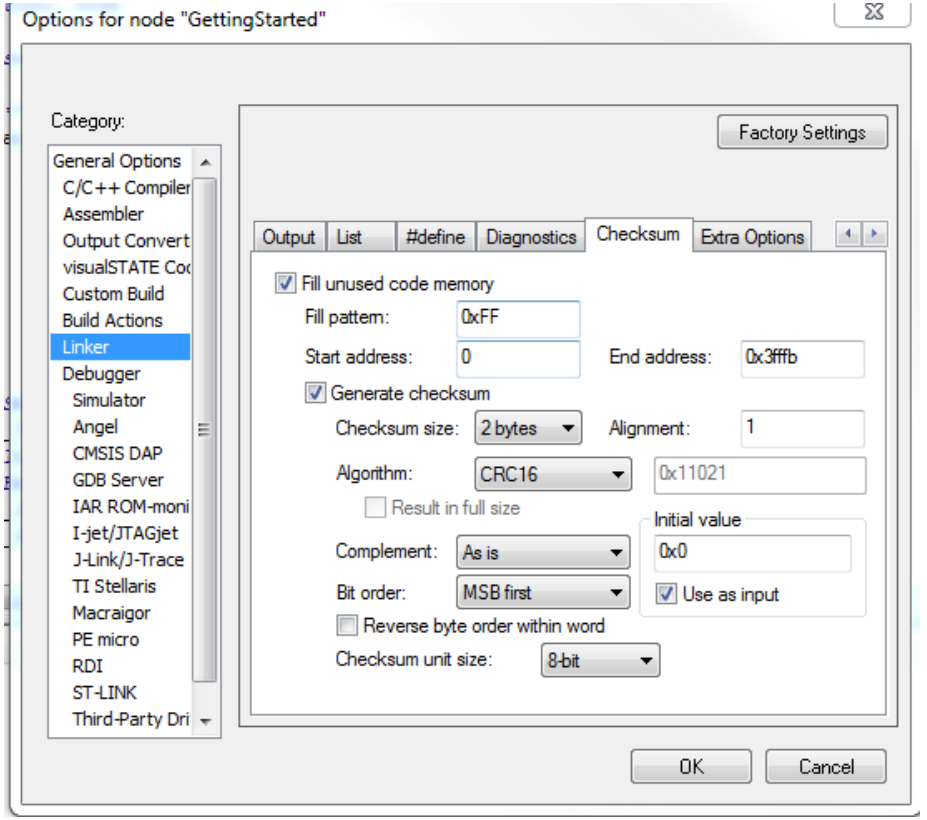
linker script (.icf file)



Linker error for check failure

Checksum calculation

- The linker can generate a checksum over the complete ROM, alternatively different sections
- After the download, the same checksum calculation is done in source code and the result is compared to verify that the flash is not corrupted
- Relevant Technical Notes:
  - Checksum calculation with IELFTOOL after linking with ILINK
    http://supp.iar.com/Support/?note=11927
  - IELFTOOL Checksum - over several ranges
    http://supp.iar.com/Support/?note=53274
  - Calculate CRC32 as in K60 hardware
    http://supp.iar.com/Support/?note=85753



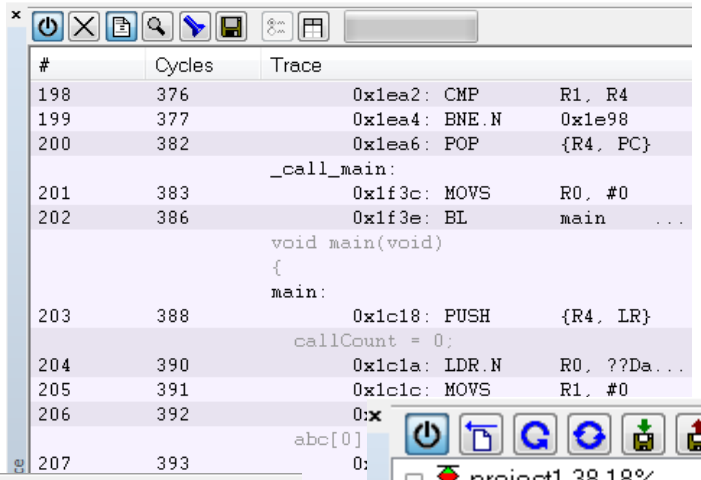
Agenda

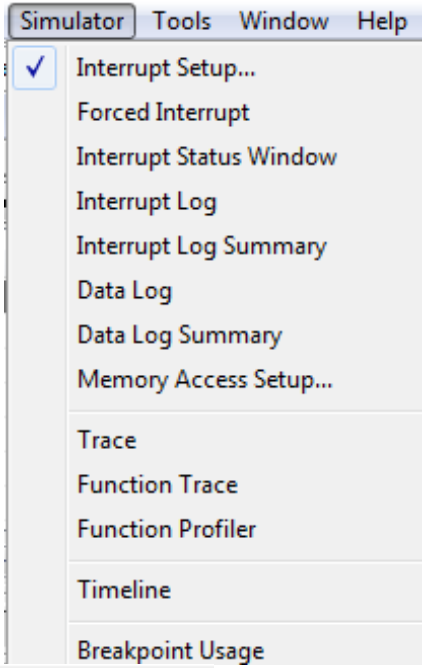
- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

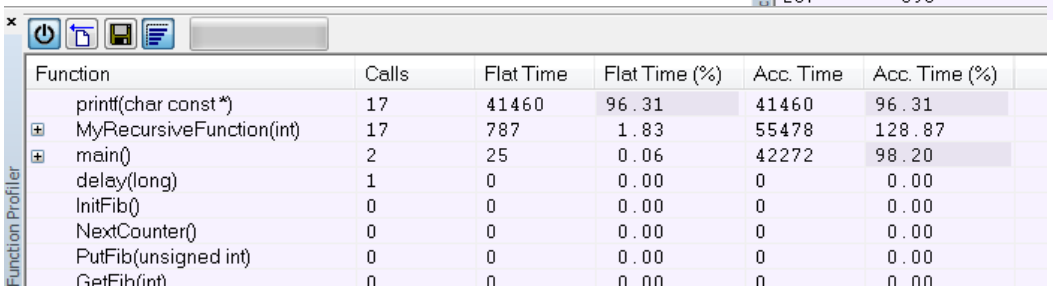
C-SPY Simulator

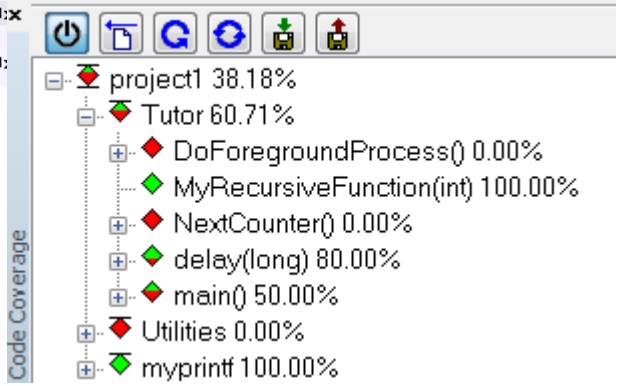
In contrast to the debugger with bandwidth-limited communication interface, the simulator does not miss anything and can show

- Complete (Function) Trace
- call graph
- code coverage



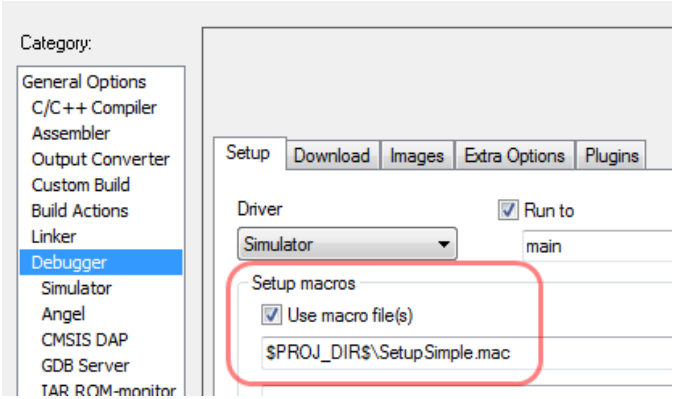






C-SPY Macros

C-SPY macros can be written to help you with complex debugging situations. There are several predefine macros that you can combine with variables, IO and other functions into new macros.



- Interrupt handling (cancel, disable, enable..)
- Reset (issue reset, different types)
- Set/remove breakpoints
- Access files
- Access data

Macros with predefined names will be called at certain times during debugging.

- execUserReset
- execUserExecutionStarted
- execUserPreReset

On-Target debugging: I-jet™

- Supports ARM7/ARM9/ARM11 and Cortex-M/R/A cores
- Hi-speed USB 2.0 interface (480Mbps)
- Target power of up to 400mA can be supplied from I-jet with overload protection
- **Target power consumption** can be measured with ~200µA resolution at 200kHz
- JTAG and Serial Wire Debug (SWD) clocks up to 32MHz (no limit on the MCU clock speed)
- Support for **SWO speeds of up to 60MHz**
- Serial Wire Viewer (SWV) with UART and **Manchester encoding**

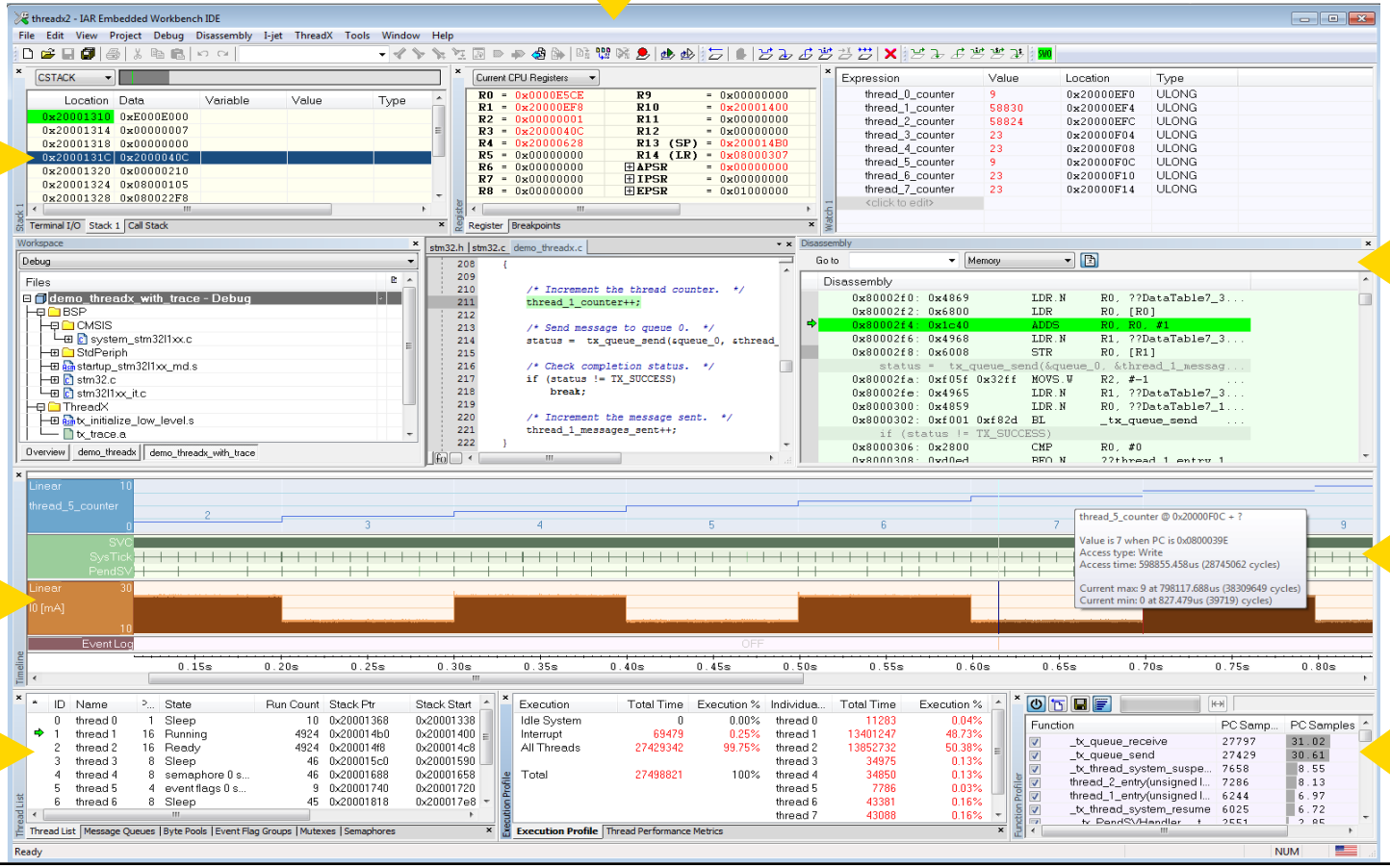
Comprehensive debugger

Integrated debugger for source and disassembly debugging

Edit source files without leaving the debug session

- Broad range of in-circuit debugging probes supported
- Trace support
- Direct flash erase and download
- C like macro system extends debugger capabilities
- Built-in simulator driver

Stack usage



Dockable windows and tab groups

Power vizualization

Timeline window

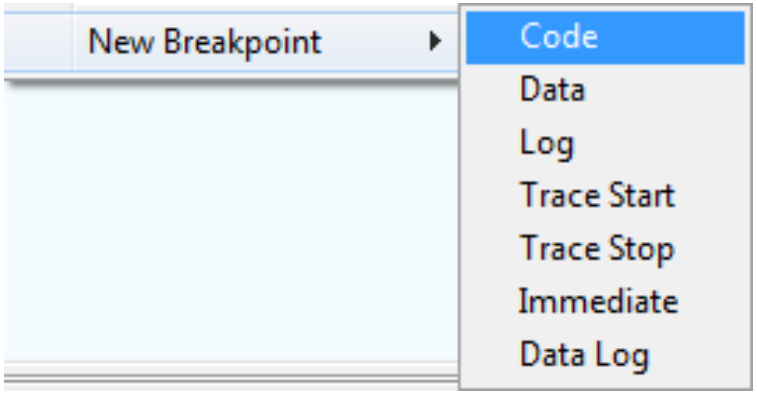
RTOS awareness

Performance analysis

Other Breakpoints

There are more than just “normal” code breakpoints:

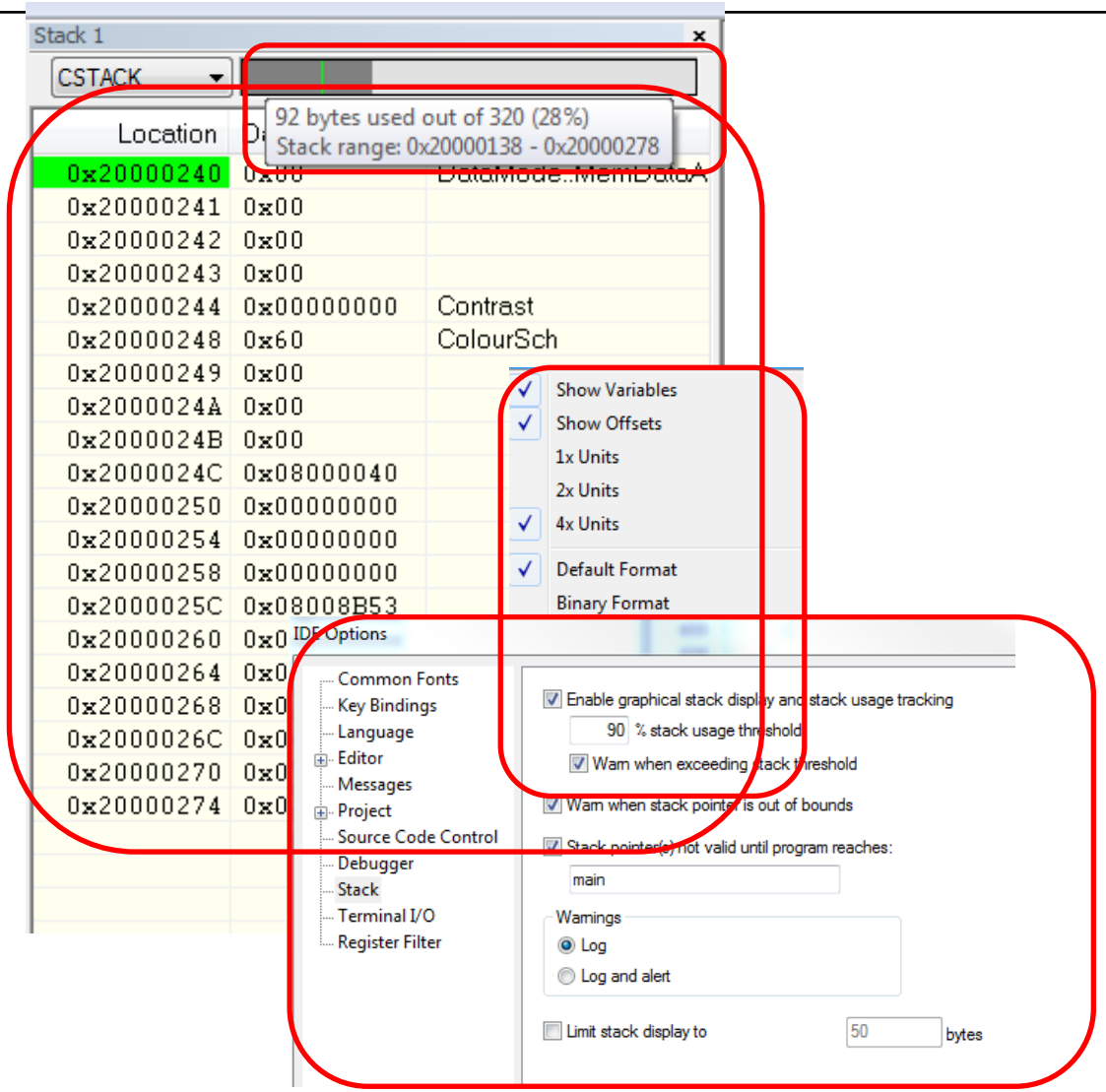
- Data breakpoint
  halts the target at access of a specific memory location/range
  set in memory window, optional with a certain pattern match (dependent on driver)
- Log breakpoint
  logs execution to the debug log window
- Immediate breakpoint
  simulator only, association with C-SPY macros
- Data Log breakpoints
  Cortex-M3 and Simulator only
- Trace Start/Stop breakpoints



Stack view

Displays of:

- Current content of stack
- Current depth of stack
- Max. depth of stack while the target was running
- Optional with/without variables or offset
- Optional with threshold warning and limited stack display



C-SPY Plugin: RTOS

- Ready-made example projects
- Context-sensitive help
- Plugins for RTOS-aware debugging

Micriμm

expresslogic

Quadros Systems Inc.

freeRTOS

WITTENSTEIN

μd Micro Digital

eForce

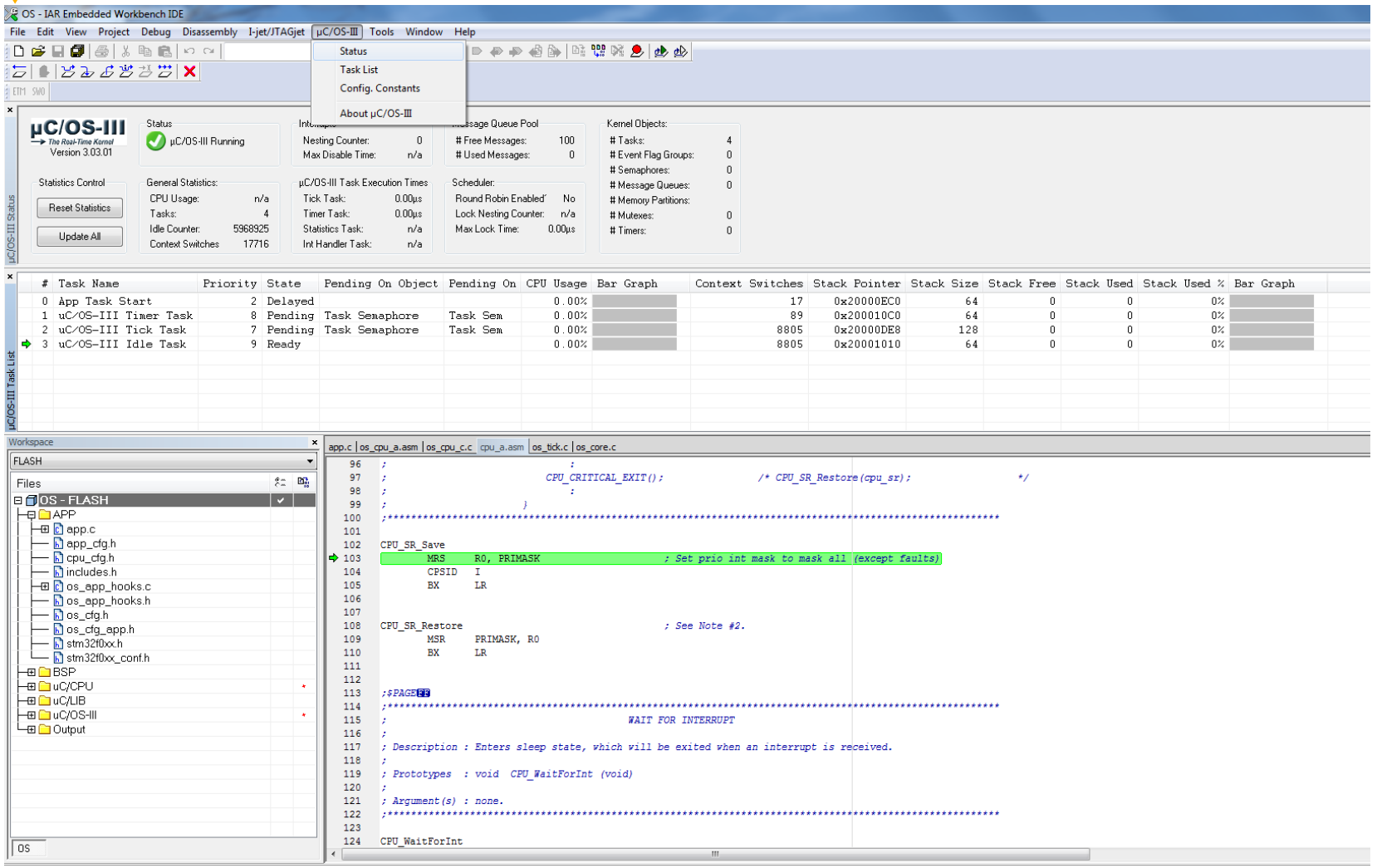
SCIOPTA

SEGGER www.segger.com

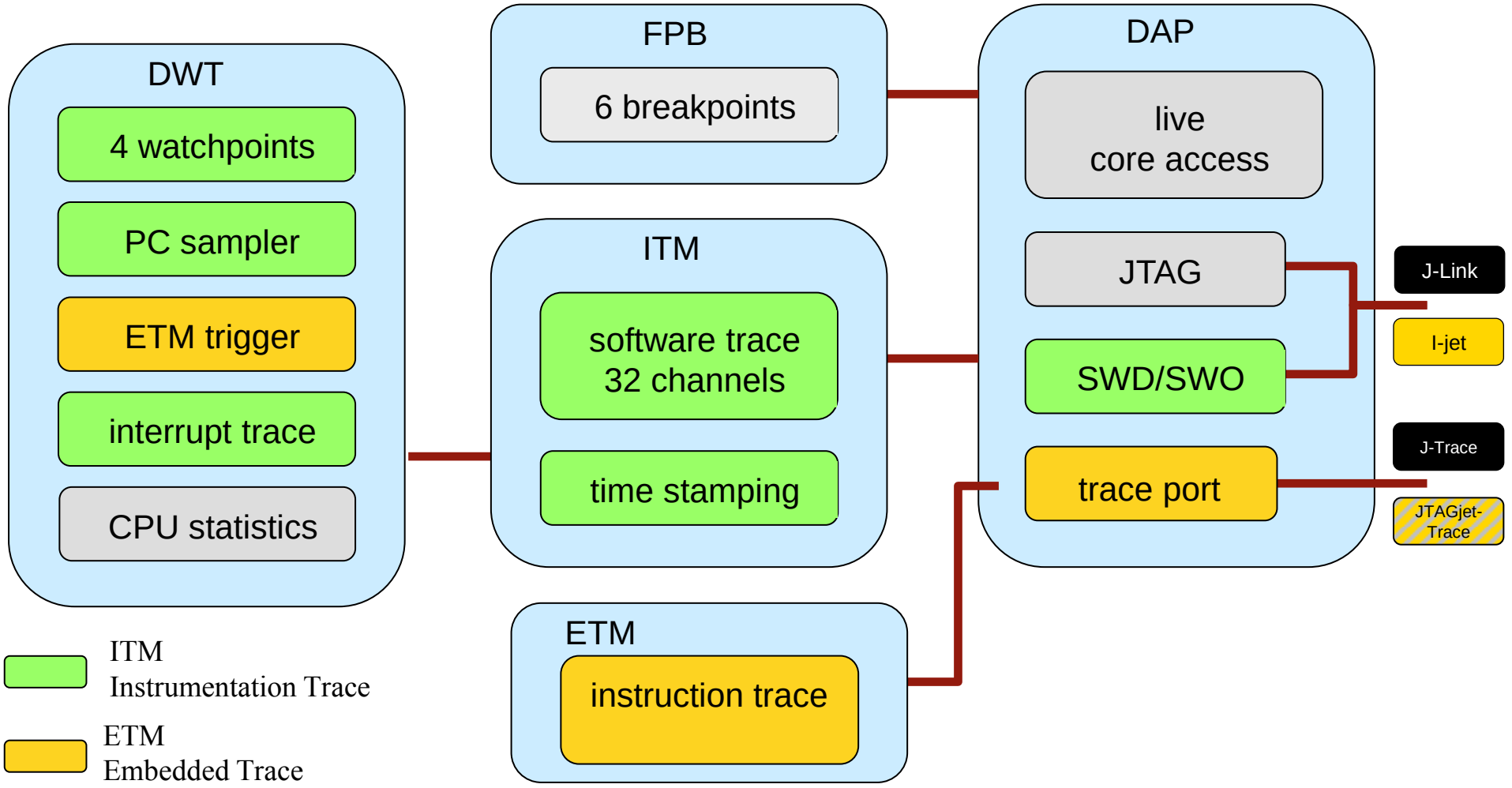
CMX SYSTEMS

freescale semiconductor

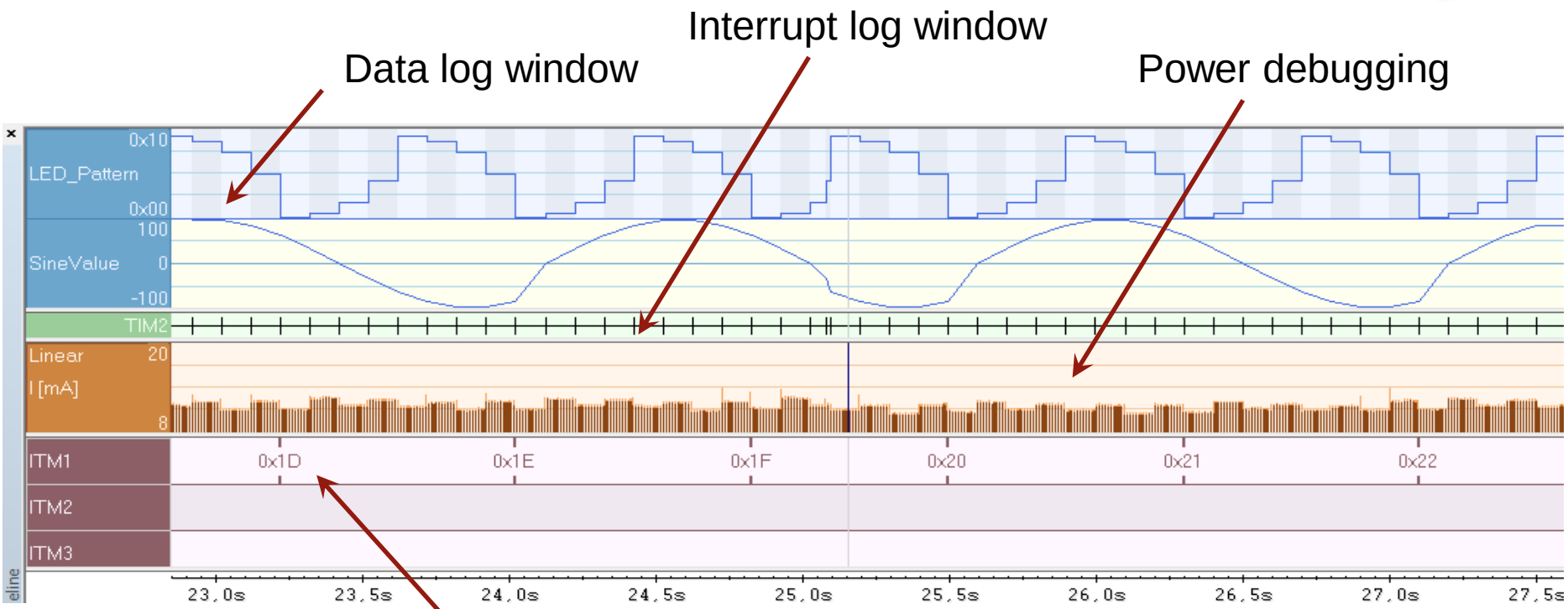
RTOS Plugins (Micrium µC/OS-III)



CoreSight Overview (ARM Cortex-M3/M4)



ITM/SWO Usage (example)



ITM event with line graph

```
ITM_EVENT8(channel_1, my_counter);
```

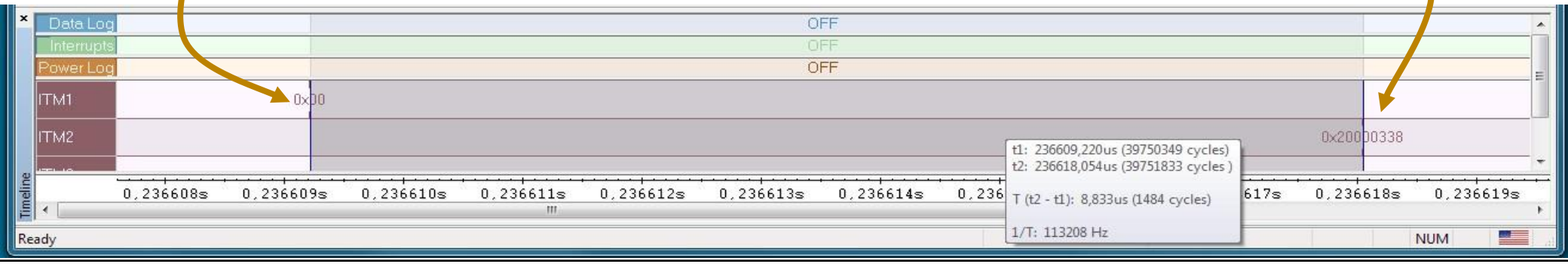
ITM events

Include the headerfile arm_itm.h, here the macros are defined.

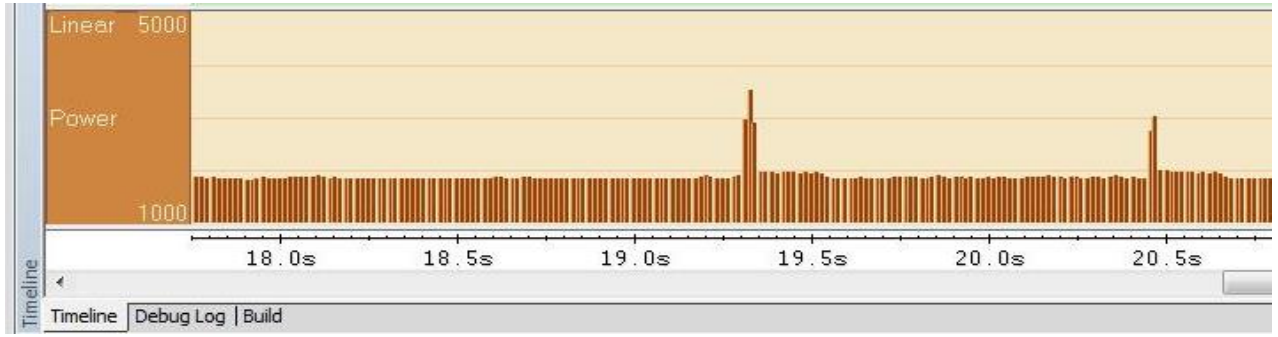
- ITM_EVENT8(channel, value)
- ITM_EVENT16(channel, value)
- ITM_EVENT32(channel, value)
- ITM_EVENT8_WITH_PC(channel, value)
- ITM_EVENT16_WITH_PC(channel, value)
- ITM_EVENT32_WITH_PC(channel, value)

- To use, simply include a macro in your code and decide what channel and what value to send
- During debugging, open the timeline window to display the data

```
ITM_EVENT8(1, 0);
I2C1_Init();
ITM_EVENT32(2,__get_SP() );
```



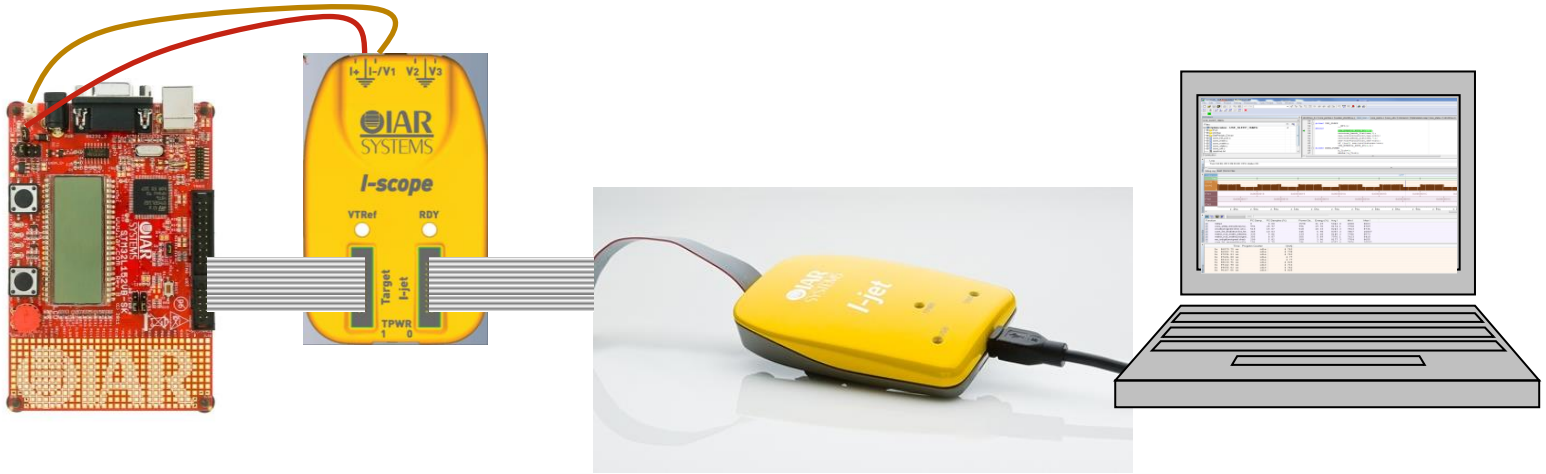
Power Debugging



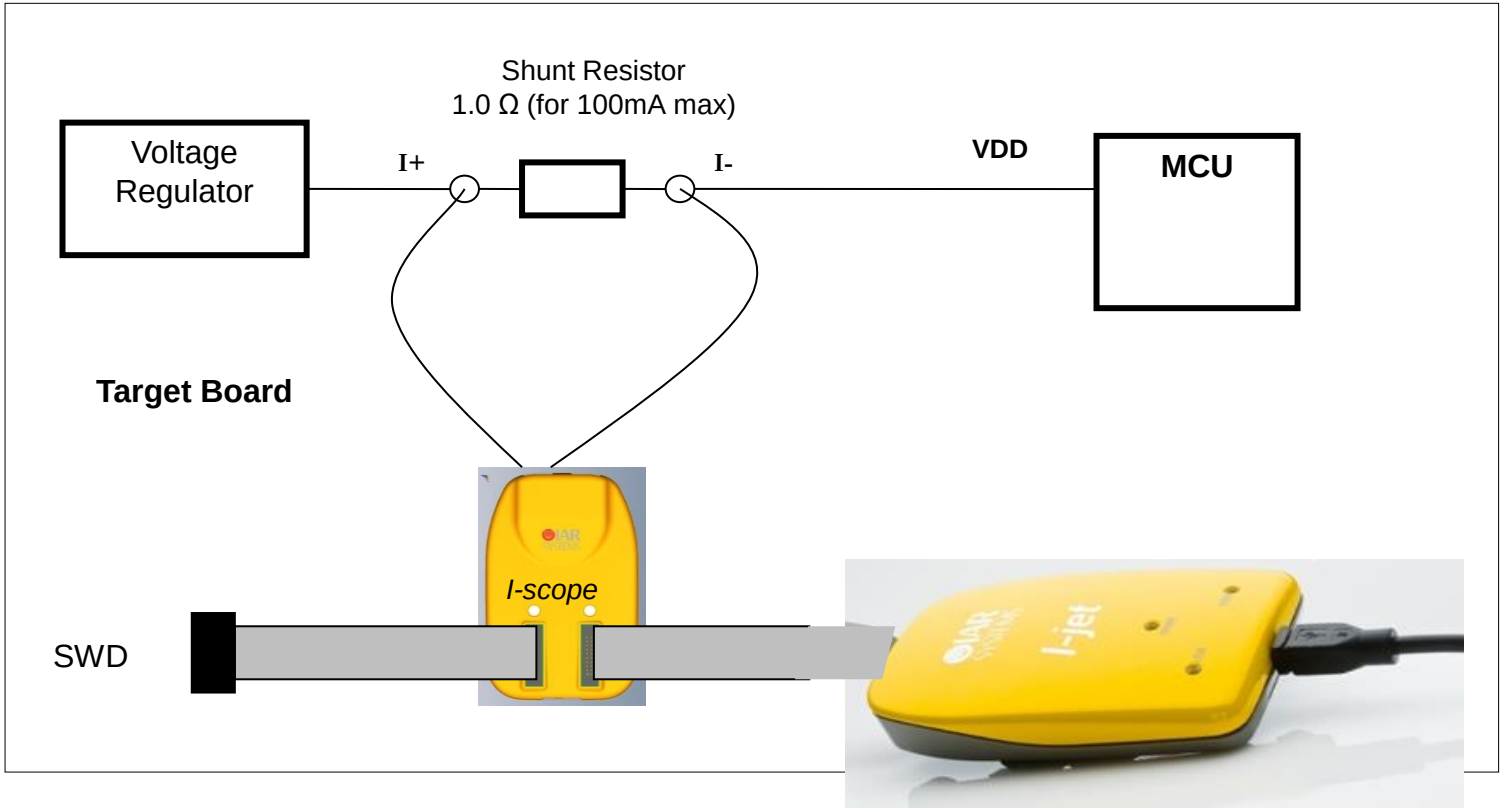
I-scope

- In-circuit measurement probe from IAR systems
- Used in combination with I-jet
- V1 – V3 voltage range 0 to 6V
- I+ and I- differential voltage, 110mV full scale
- Sampling rate up to 200 kHz, 12 bit resolution

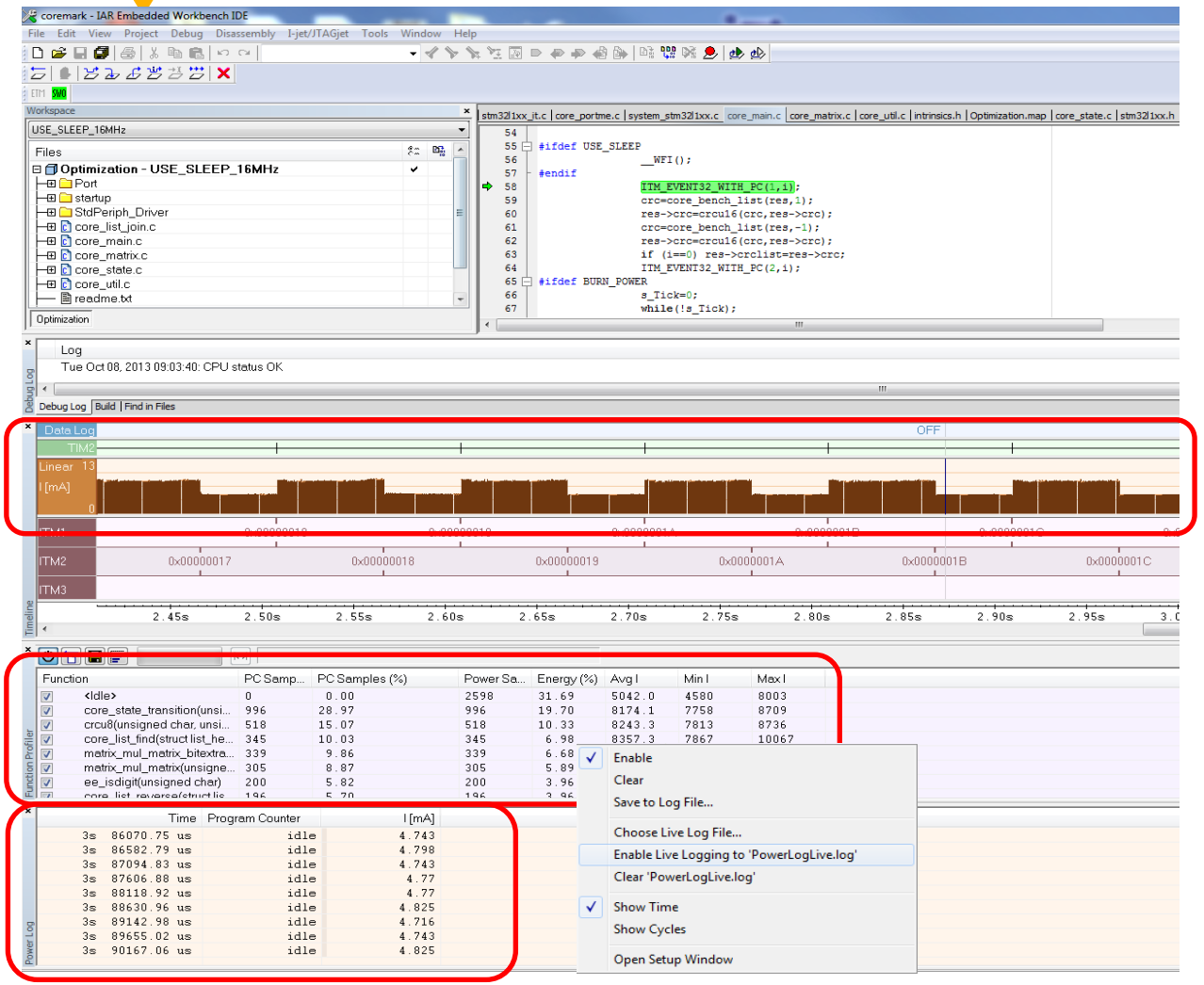
Current-/ voltage measurement



Power-Debugging

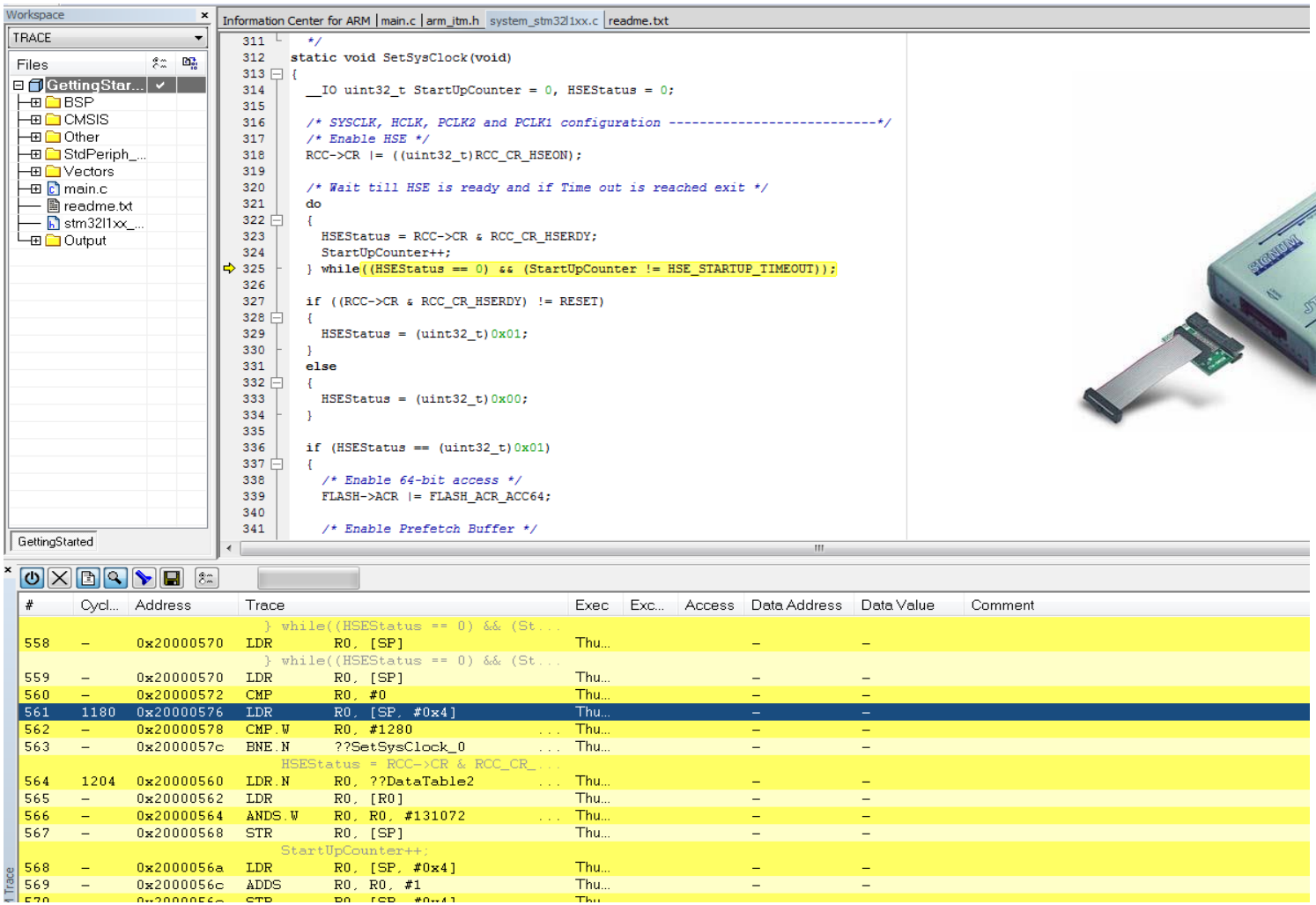


Visualizing current consumption



- In the timeline, together with other parameters
- In the function profiler
- As log
  optionally live log

JTAGjet™-Trace for Cortex-M



ETM Trace features

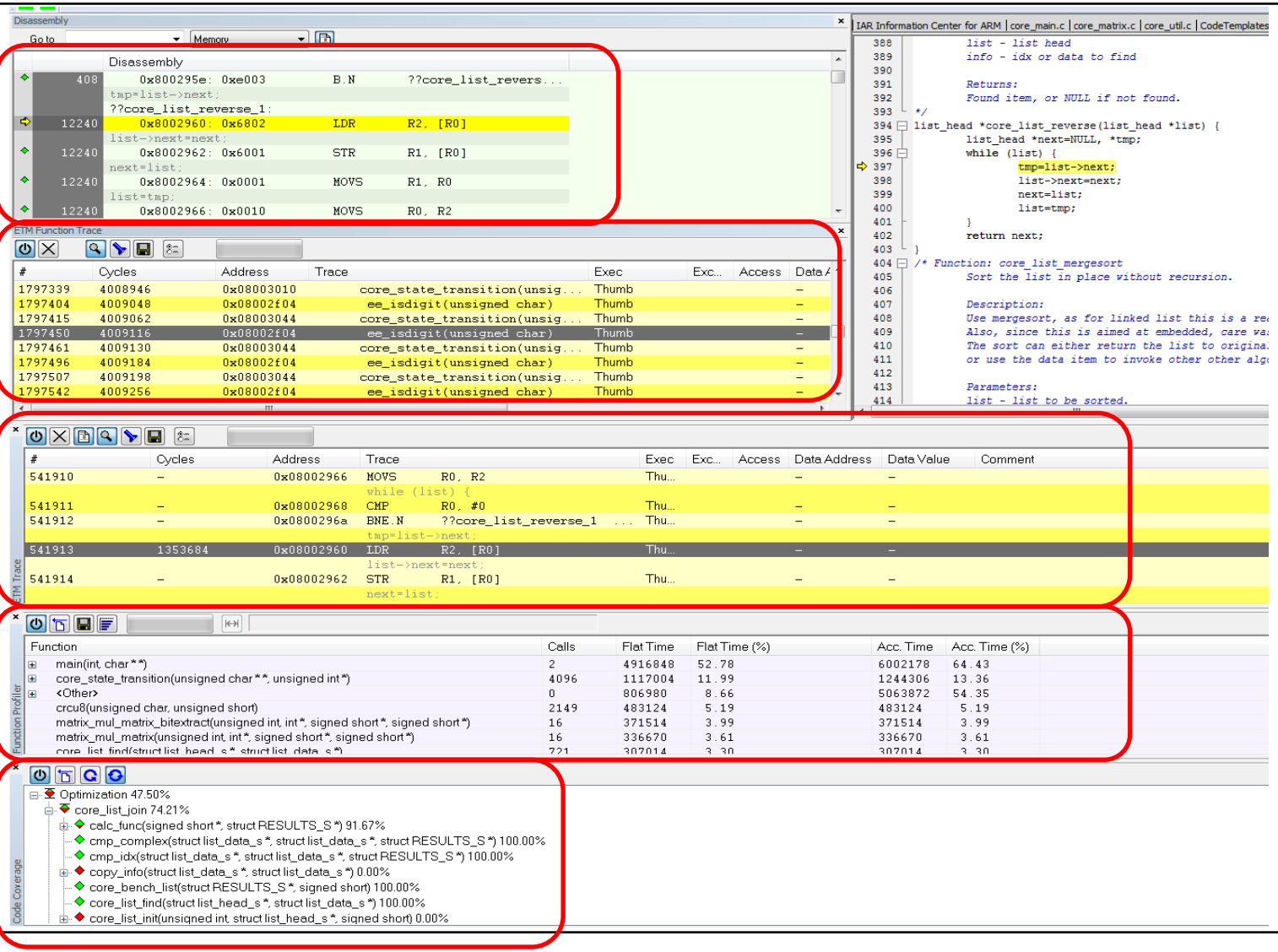
Instruction Profiling →

Function Trace →

ETM Trace →

Function Profiler →

Code Coverage →



JTAGjet™-Trace for Cortex-M

- Supports Cortex-M cores equipped with ETM trace logic
- Up to 200MHz trace clock (400Msample/sec ETM trace acquisition speed)
- Auto adjusting timing eliminates problems with data skew
- Available with 4.5 or 18 MB trace buffer
- Support for 4, 8 and 16-bit wide trace busses
- 56-bit time stamp with CPU cycle accuracy for timing analysis
- Easy access to all ETM modes, triggers and filtering
- Only one connection to target–both JTAG and trace are taken from a high-density, 20-pin Cortex header

I-jet Trace for Cortex-M coming soon!

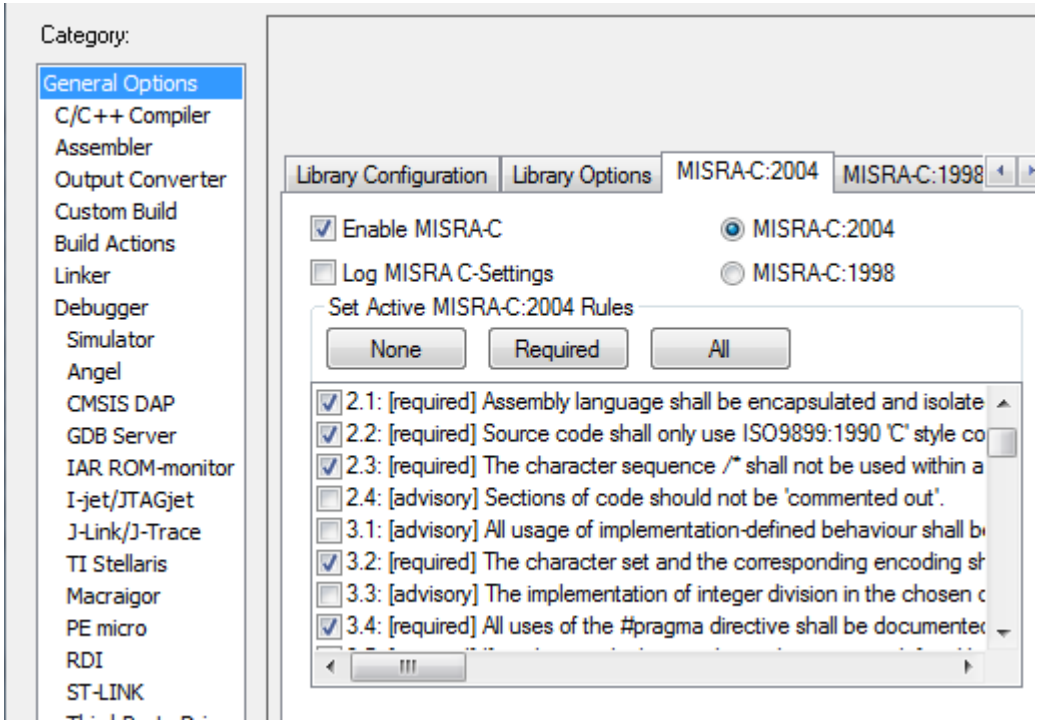
- USB powered
  - optional external power supply for higher target power
- USB 3.0 high-speed interface
  - USB 2.0 compatible
- 64/256 Mb trace memory
- Interface to I-scope

Agenda

- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

Safety – MISRA-C

- MISRA-C:2004 Checker included



Support for MISRA-C:2012 planned for 2014

Skip MISRA

MISRA C definition

MISRA C stands for the Motor Industry Software Reliability Association, a consortium based out of Cambridge in the UK that promotes standards to improve the safety and reliability of embedded code. More importantly, MISRA-C:

- Aims to facilitate code safety, portability and reliabilty
- Must be used in some cases
- Often good to use even when not mandatory
- Built in to Embedded Workbench, so it is simple to use
- Recommended in IEC61508
- Has evolved as a widely accepted model for best practices by leading developers in various sectors

Easy MISRA rules to follow

- The basic types of `char`, `int`, `short`, `long`, `float`, and `double` should not be used, but specific-length equivalents should be typedef'd for the specific compiler, and these type names used in the code (MISRA-C:2004 Rule 6.3)
  - The size and signedness of these types is different on different MCUs
  - Instead, use types like `uint16_t` that explicitly tell other developers the size and signedness you intend
- Loops and conditional statements are required to have code blocks encapsulated in { } (MISRA-C:2004 Rule 14.8)

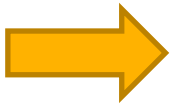
  if (exp) then { statement; } else { statement; }
  - Code intended to execute with those statements is explicit
- Implicit conversions are not allowed (MISRA-C:2004 Rule 10.1)
  - *uint16_t i = 1; /* ? */*

MISRA C 2004 Rule 12.4

This rule states:

the right hand operand of an && or || operator shall not contain side effects ()

```
if((a==b)&&(c--))
{
      ...
}
```



```
if((a==b)
{
        if(c--))
        {
              ...
        }
}
```

MISRA C 2004 Rule 19.10

This rule states:

In the definition of a function-like macro, each instance of a parameter shall be enclosed in parenthesis ()

Example:

```
#define TIMES_TWO(x) x*2
a=TIMES_TWO(2+3);
```

MISRA C 2004 Rule 19.10

This rule states:

In the definition of a function-like macro, each instance of a parameter shall be enclosed in parenthesis ()

Example:

```
#define TIMES_TWO(x) x*2
a=TIMES_TWO(2+3);    //    2+3*2 = 8 ≠ 10
```

MISRA C 2004 Rule 19.10

This rule states:

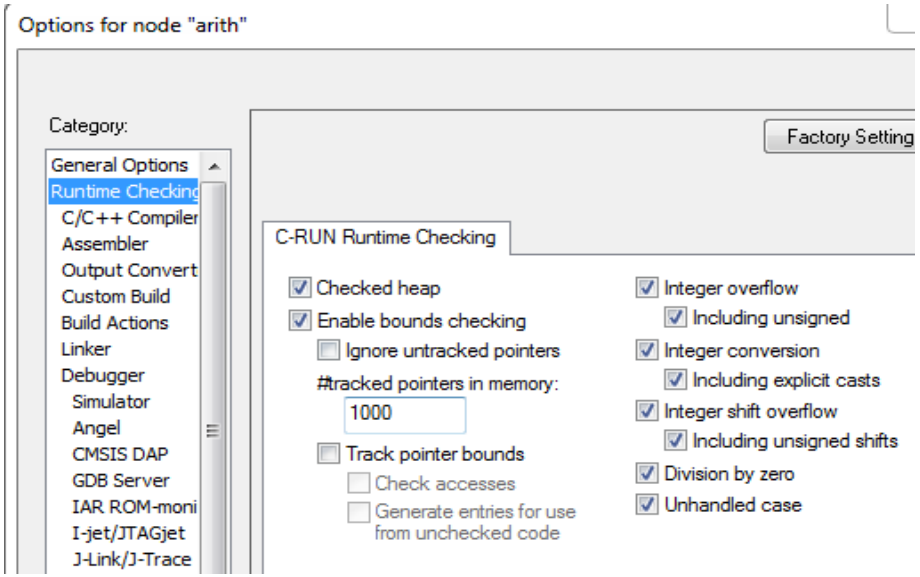
In the definition of a function-like macro, each instance of a parameter shall be enclosed in parenthesis ()

Example:

```
#define TIMES_TWO(x)(x)*2
a=TIMES_TWO(2+3);    //     (2+3)*2 = 10
```

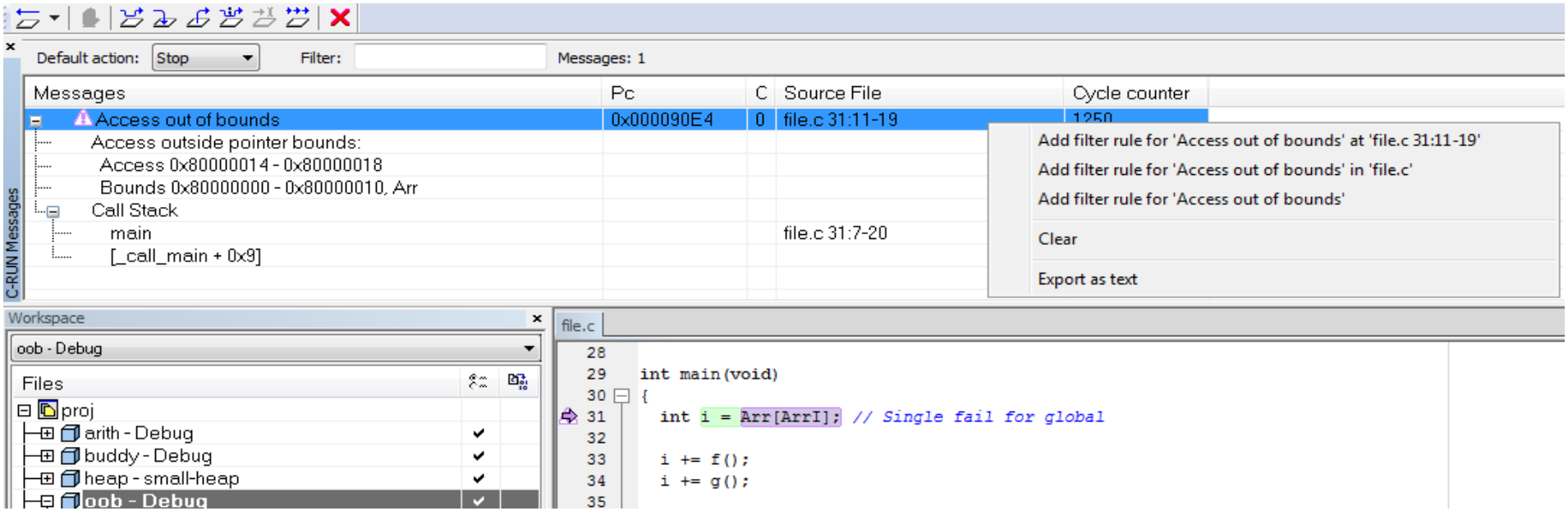
C-RUN Runtime Analysis

- ***Available in***
  IAR Embedded Workbench for ARM
  now
- ***Individual checks can be turned on/off***
- ***Possible to configure on module basis***
- ***Overhead depends on***
  - ***Performed checks***
  - ***Application profile***



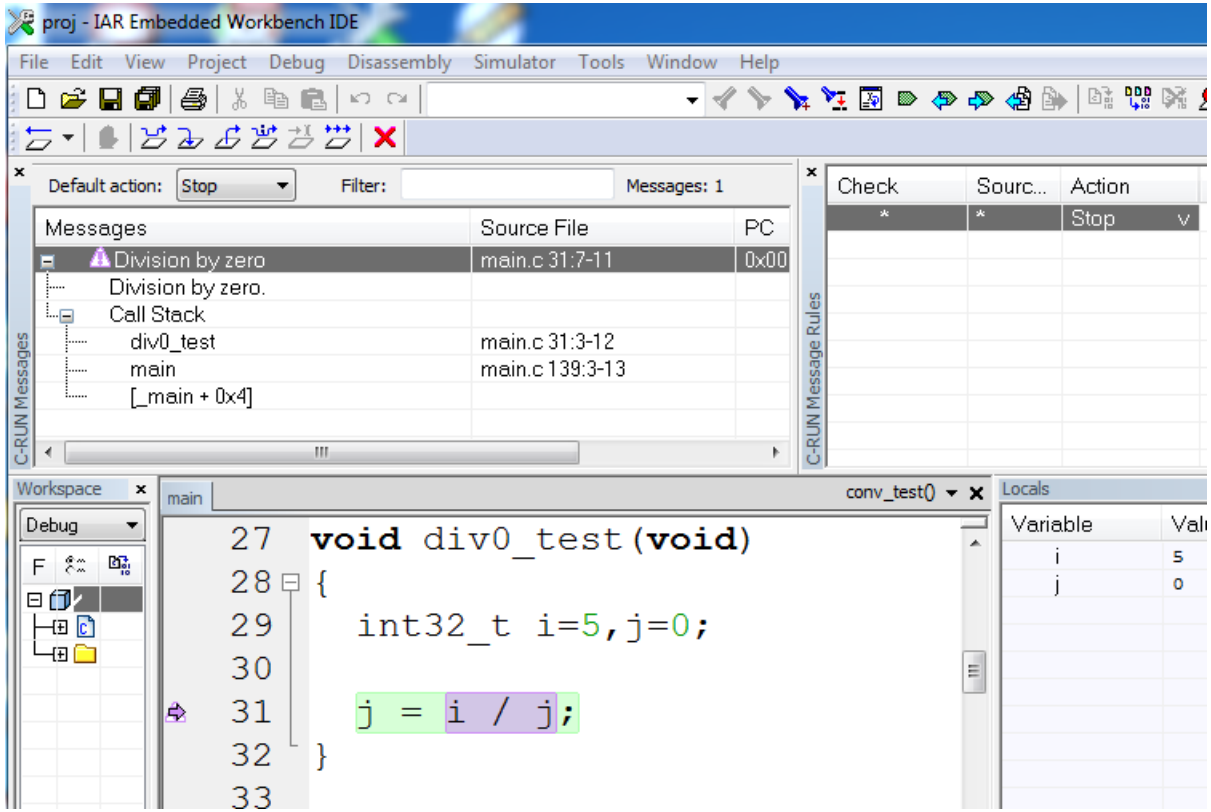
C-RUN Runtime Analysis

- ***Fully integrated in IAR Embedded Workbench***
- ***Immediate feedback in the daily code-and-test cycle***
- ***Very detailed feedback on what went wrong***
- ***Flexible rules to treat messages on various code levels***



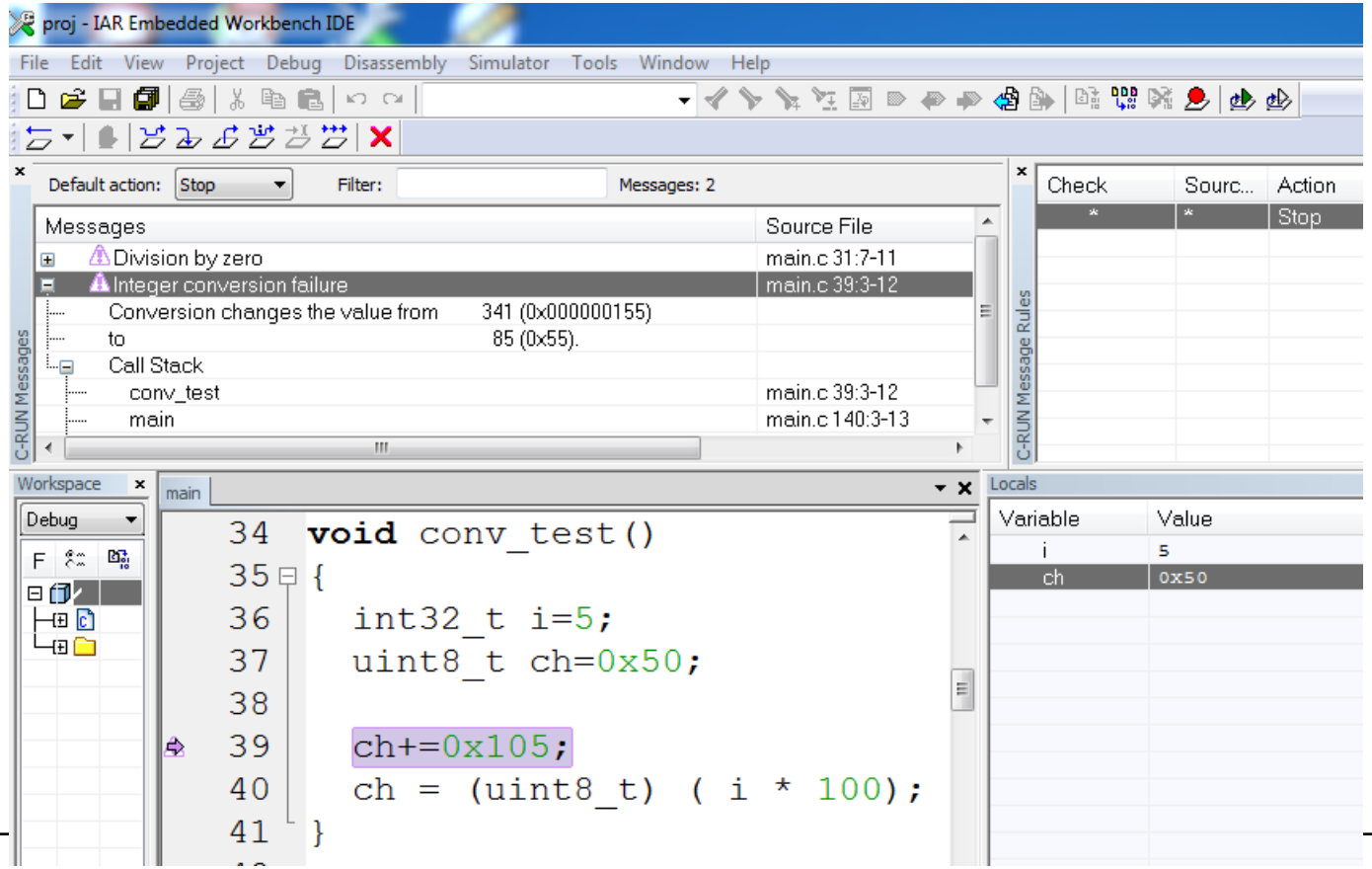
C-RUN Runtime Analysis

- Very efficient instrumentation code
- Very detailed feedback on what went wrong



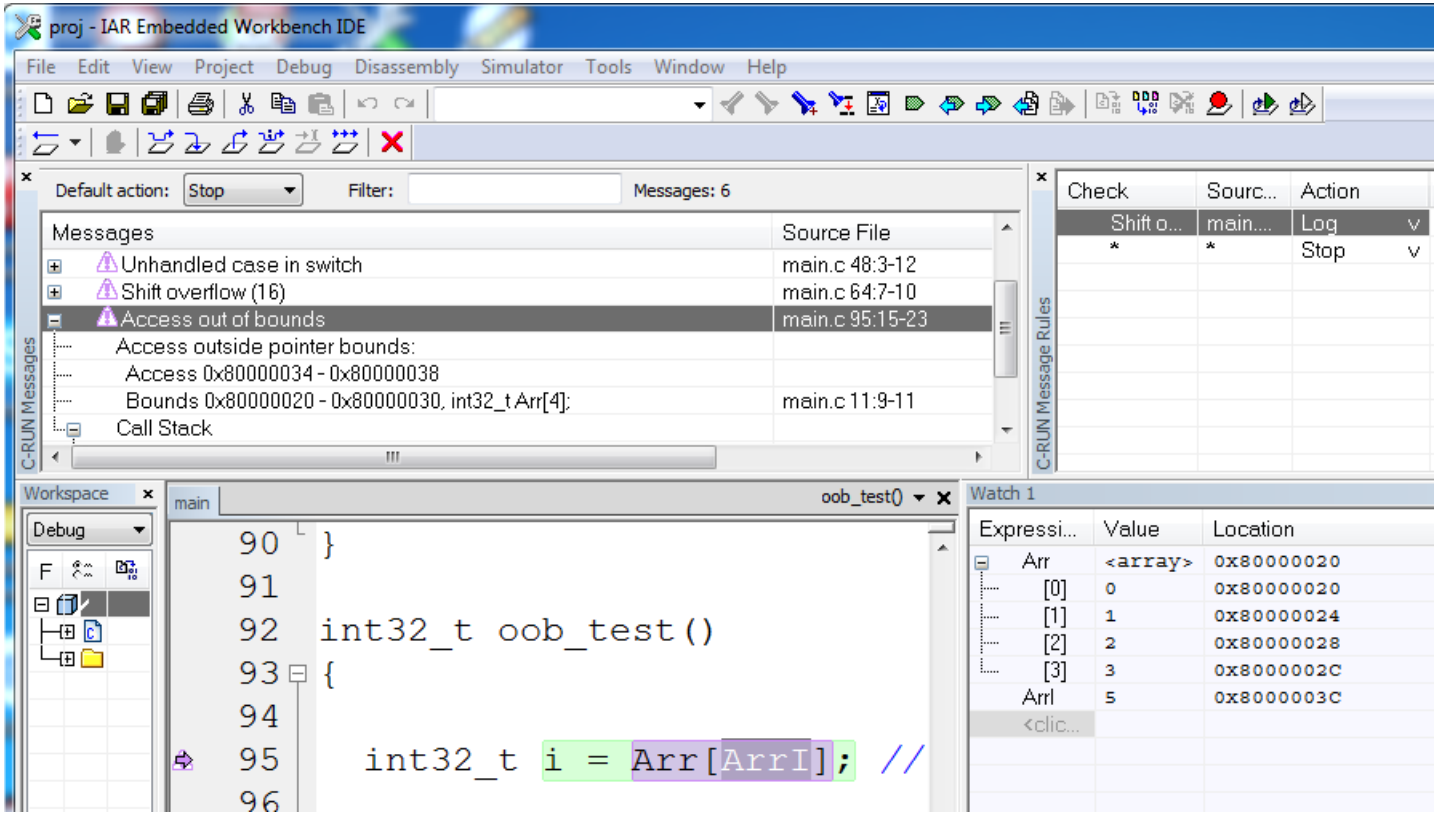
C-RUN Runtime Analysis

- Very efficient instrumentation code
- Very detailed feedback on what went wrong



C-RUN Runtime Analysis

- Very efficient instrumentation code
- Very detailed feedback on what went wrong



Validation of Compliance

- Document describes test methods at IAR Systems
- provided on request free of charge for required release
- Very useful for certification purposes

IAR Systems External	Document number: 21559	Page 1(6)
Prepared By Anders Lundgren	Date 2014-03-03	Revision 1

IAR Embedded Workbench for ARM 7.10 Statement of standards compliance

Validation of compliance document

Contents

EWARM – TÜV certified tool chain

IAR SYSTEMS

ZERTIFIKAT ◆ CERTIFICATE ◆ 認証書 ◆ СЕРТИФИКАТ ◆ CERTIFICADO ◆ CERTIFICAT

Product Service

CERTIFICATE

No. Z10 13 04 84282 001

Holder of Certificate: IAR Systems AB
Strandbodgatan 1
750 23 Uppsala
SWEDEN

Factory(ies): 84282

Certification Mark:

Product: Software Tool for Safety Related Development

Model(s): IAR Embedded Workbench for ARM
- Build Tool Chain

Parameters: The development tool chain fulfills the requirements for development tools classified T3 according to IEC 61508-3. The build tool chain is qualified to be used in safety-related software development according to IEC 61508 and ISO 26262.
The report IU84911C is a mandatory part of this certificate.

Tested according to: IEC 61508-3:2010
ISO 26262-8:2011

The product was tested on a voluntary basis and complies with the essential requirements. The certification mark shown above can be affixed on the product. It is not permitted to alter the certification mark in any way. In addition the certification holder must not transfer the certificate to third parties. See also notes overleaf.

Test report no.: IU84911C

Date, 2013-04-04
Page 1 of 1

(Guido Neumann)

472452

TÜV SÜD Product Service GmbH · Zertifizierstelle · Ridlerstraße 65 · 80339 München · Germany

TÜV®

A1 / 04.11

Report

to the

Certificate

Z10 13 04 84282 001

IAR Embedded Workbench for ARM build tool chain

Manufacturer:
IAR Systems AB
Strandbodgatan 1
Uppsala
Sweden

Report no. IU84911C
Revision 1.0 of April 8th, 2013

Test and Certification Body

TÜV SÜD Rail GmbH
Generic Safety Systems
D-80339 Munich

page 1 of 10

Agenda

- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

IAR Academy

- Getting started
  - IAR Embedded Workbench overview
  - Compiler
  - Assembler
  - Linker
  - Configuration
  - Debugger and Simulator
- Efficient programming
  - Compiler technology
  - Coding techniques
  - Best practices
  - Linking applications
  - Safety and standards
- Advanced debugging (Cortex-M3/4 focus)
  - Advanced debugging
  - Power debugging
  - Using trace
- Customized Training
  - On Request

Details and registration:
www.iar.com/academy

Nächster Termin: 9.- 10. Juli in München

Agenda

- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

Exercises and Demo