

Overview IAR Embedded Workbench for ARM

Infineon XMC 45xx TechDay

Munich, June 25 2014

Milan, July 1 2014

Martin Gisbert
FAE IAR Systems



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

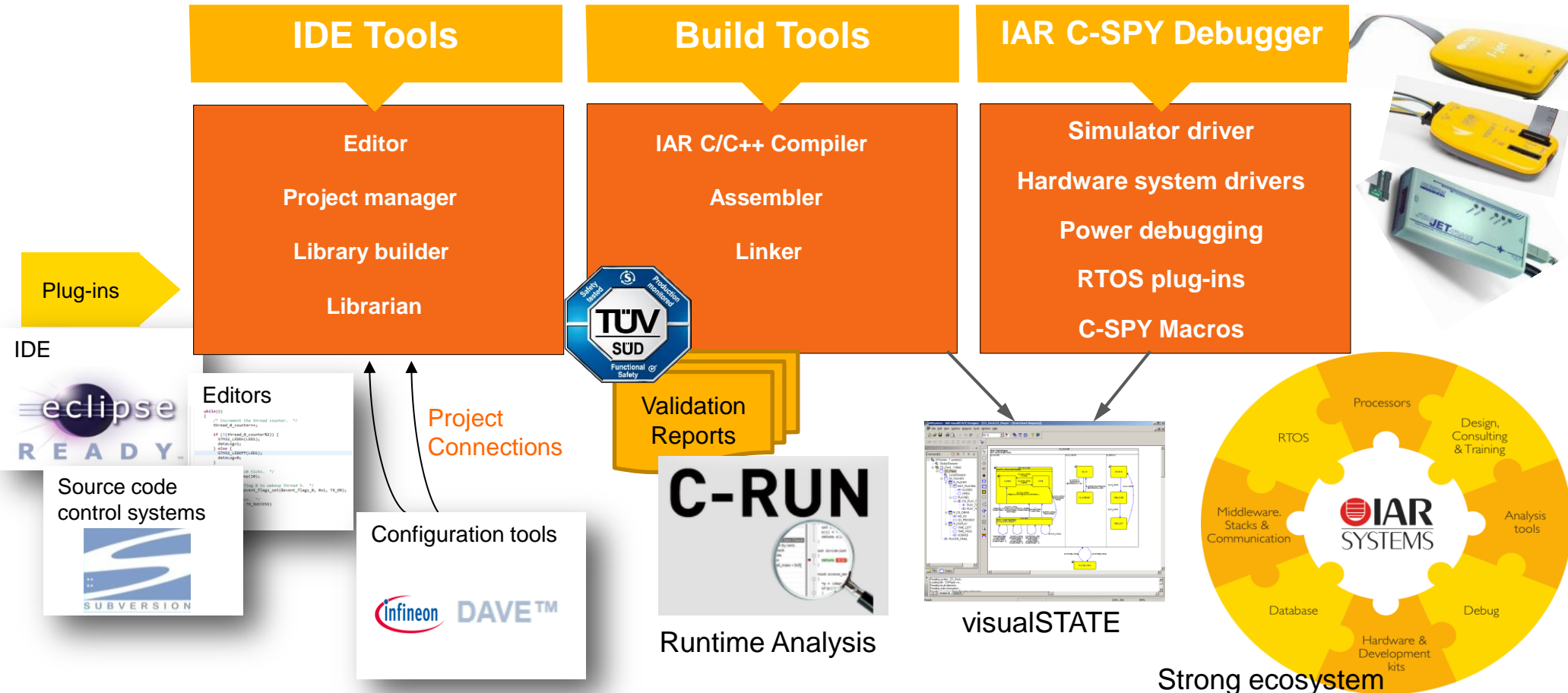


• Overview

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

What is IAR Embedded Workbench?

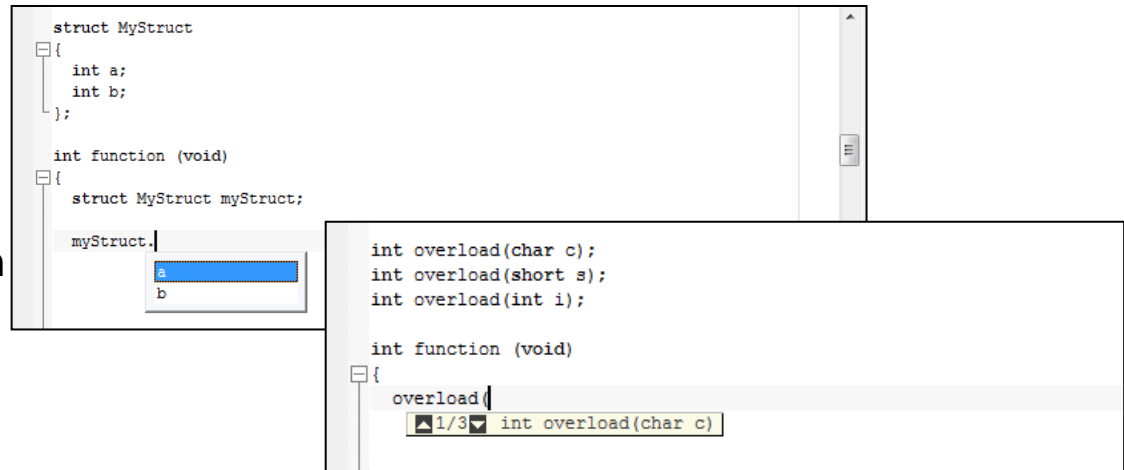
IAR Embedded Workbench IDE



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

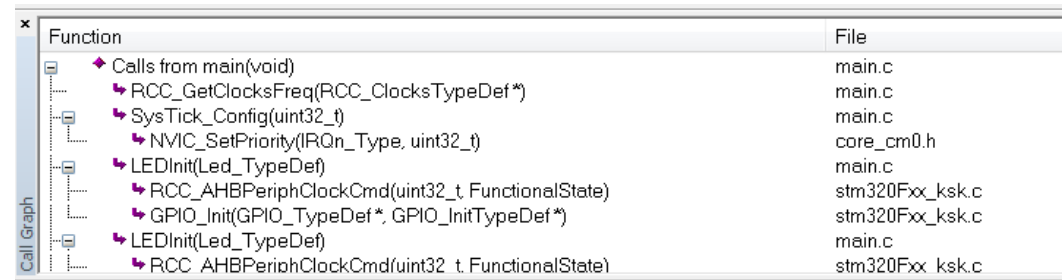
- Improved Editor with

- code and comment folding
- Word and Code completion
- Parameter Hint

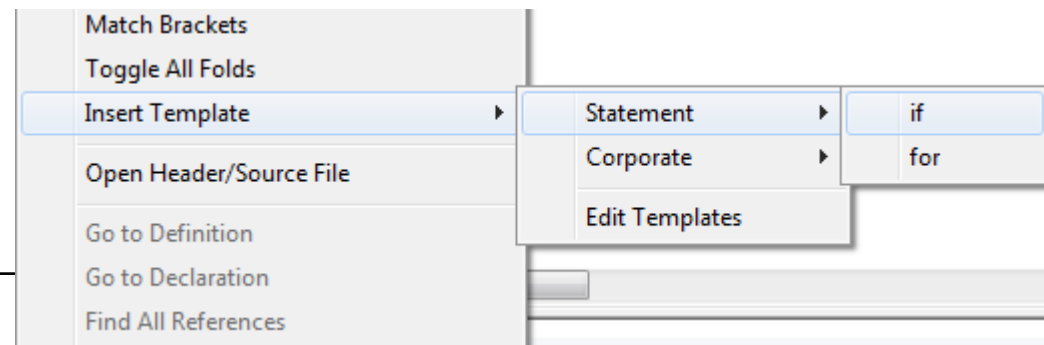


- Source Browser

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



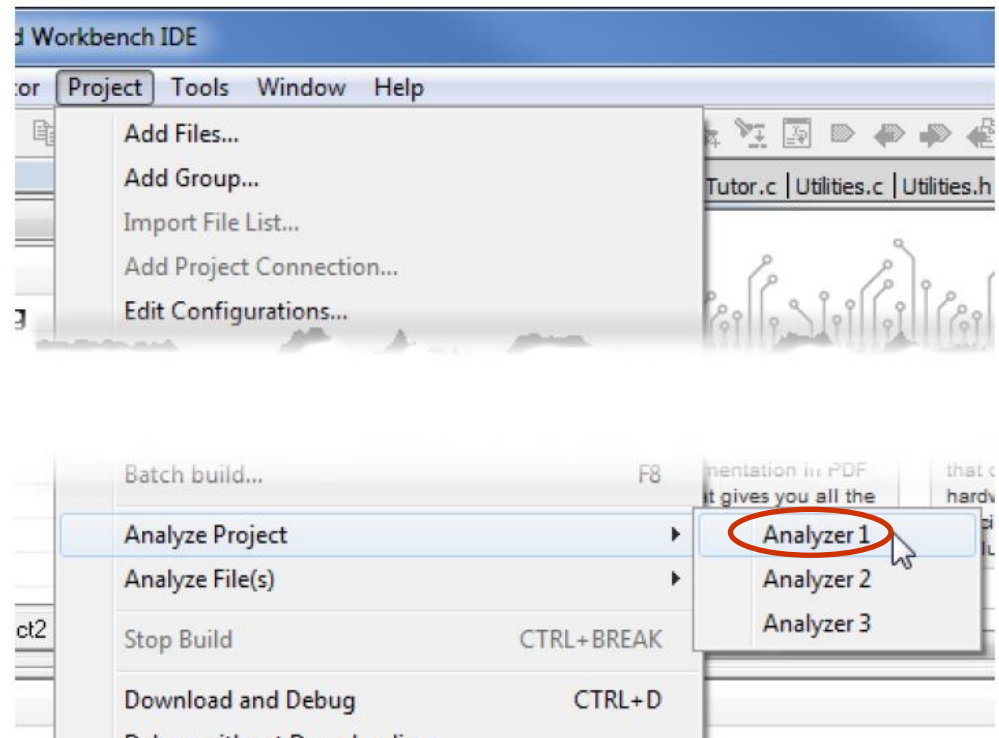
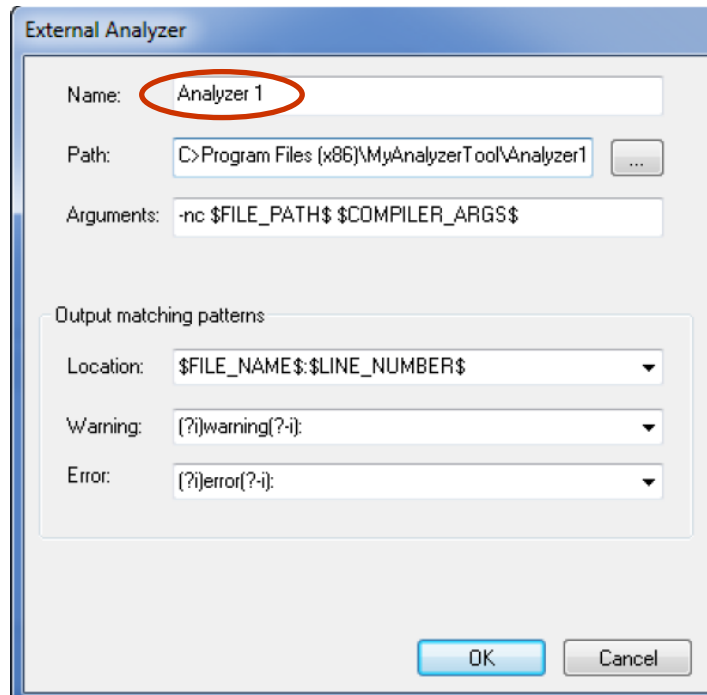
- Code templates



- 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 below.

Requirements

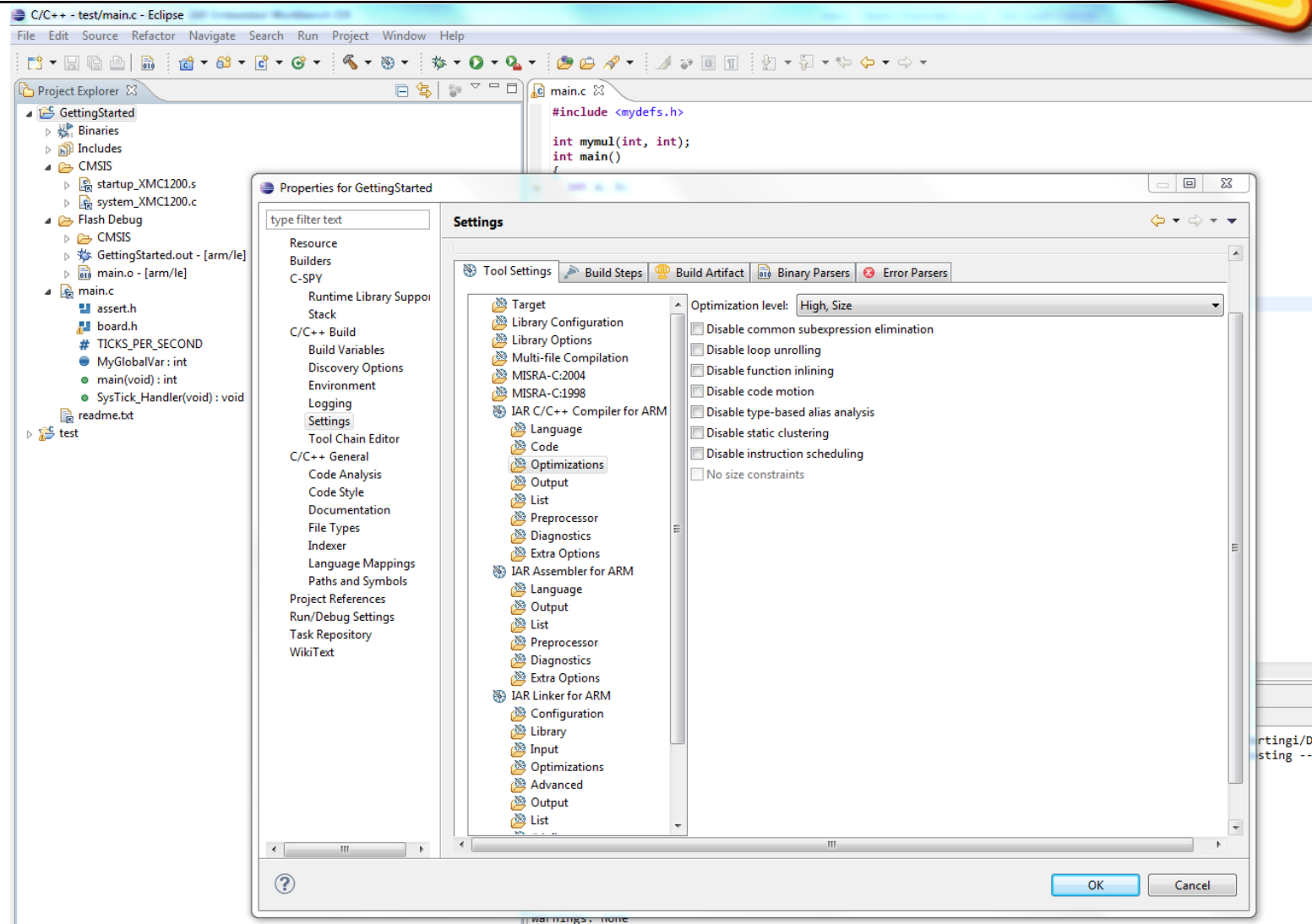
- Eclipse. [Download Eclipse](#).
- IAR Embedded Workbench (the IAR Eclipse plugins do not include the compiler or debugger itself).
- Java Runtime Environment, version 6 or later. [Download Java](#).

ARM

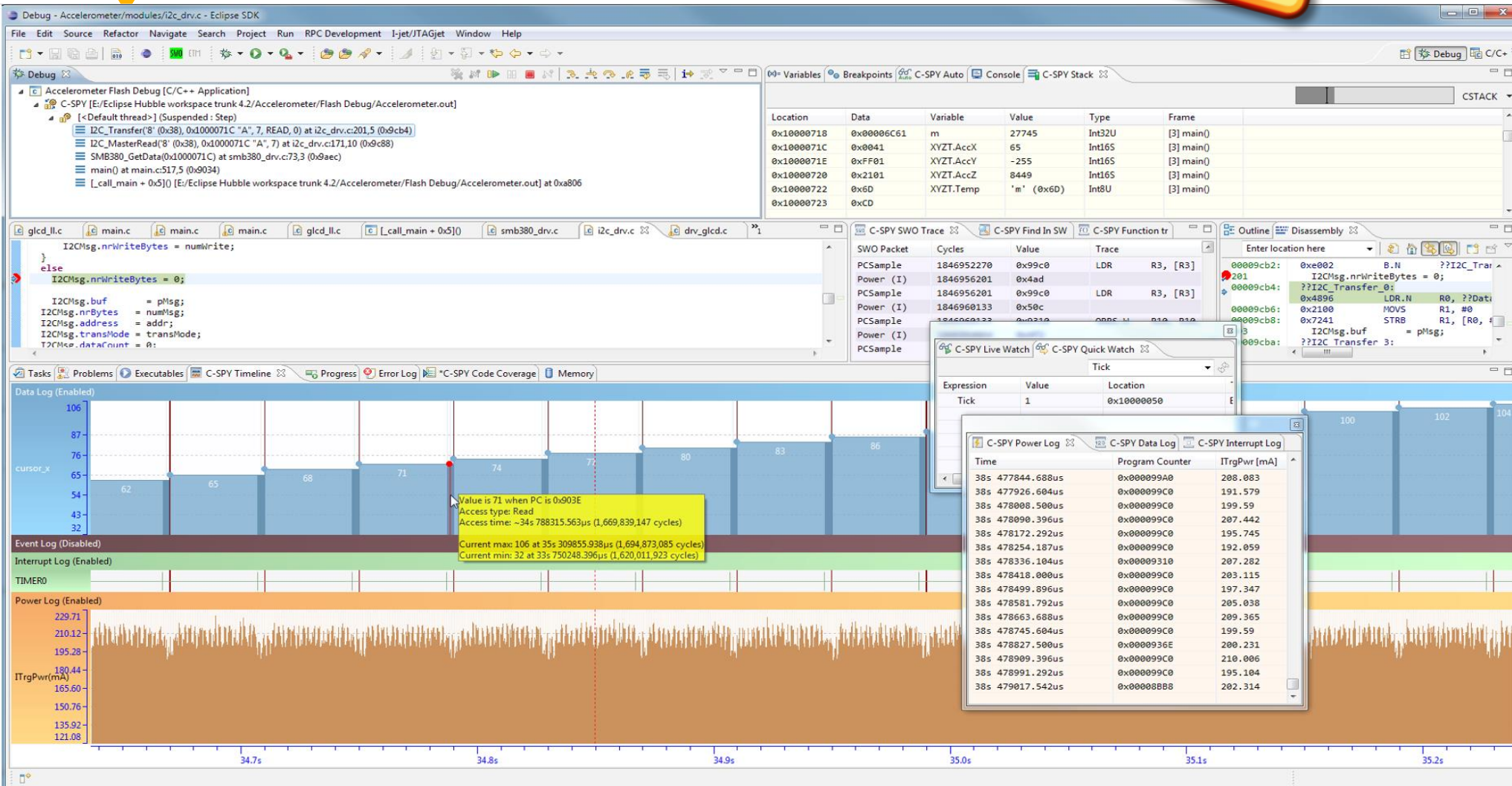
Includes build and debug support.

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)

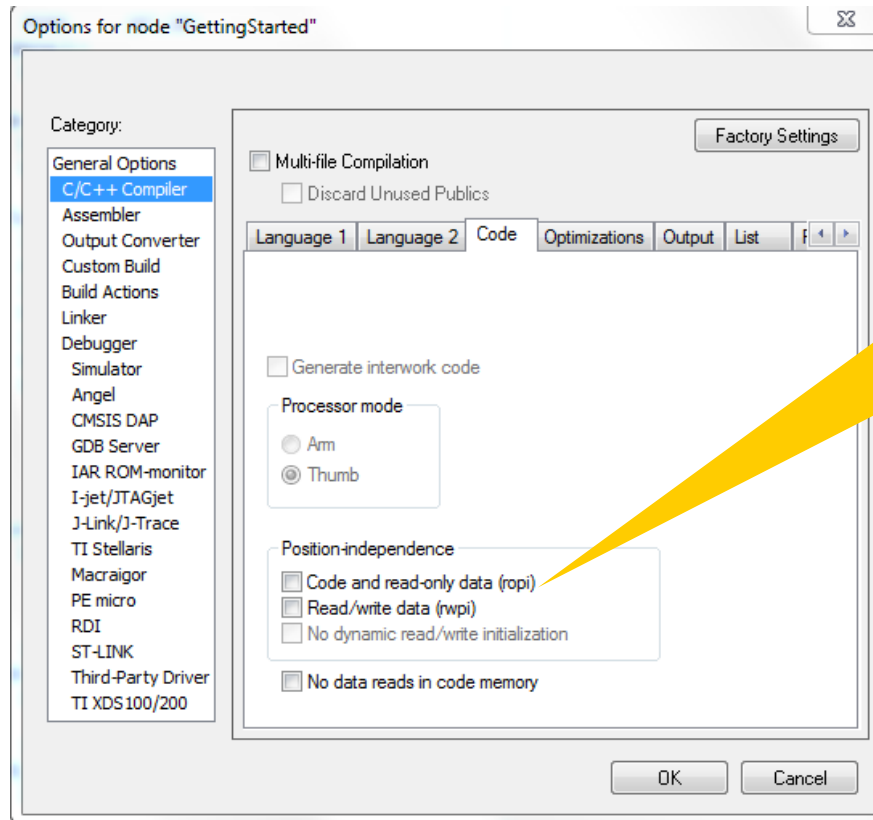


Eclipse Plugin (Debugging View)



- 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

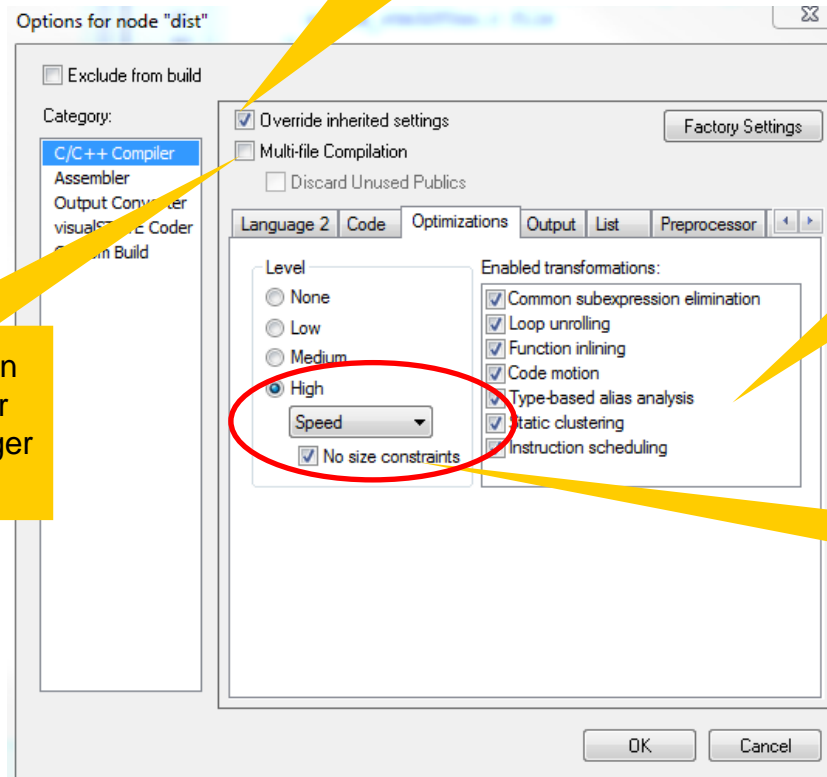
Powerful C/C++ Compiler



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

Major functions of the optimizer can be controlled individually

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



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???

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

Placement controlled via text file

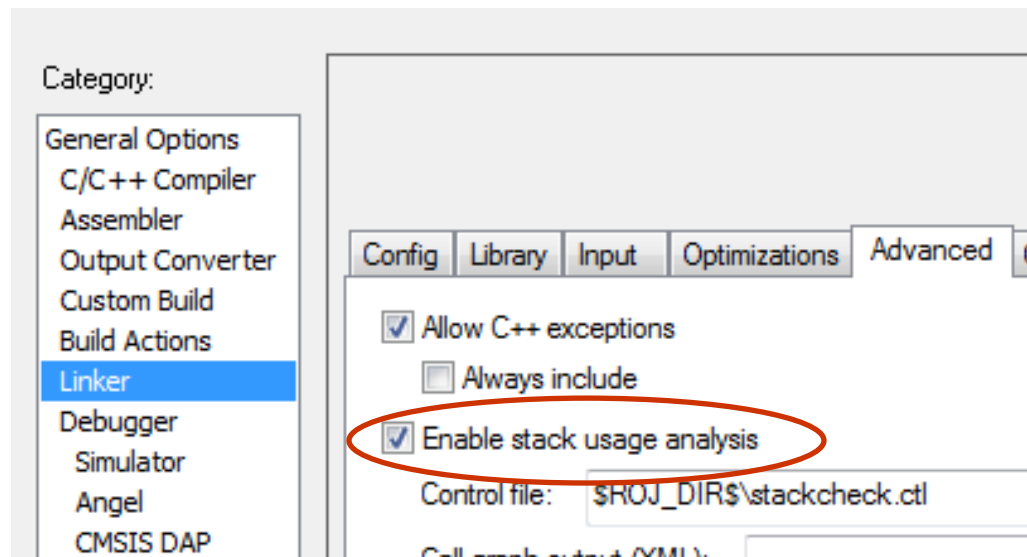
IlLink supports several types of section selectors:

- 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

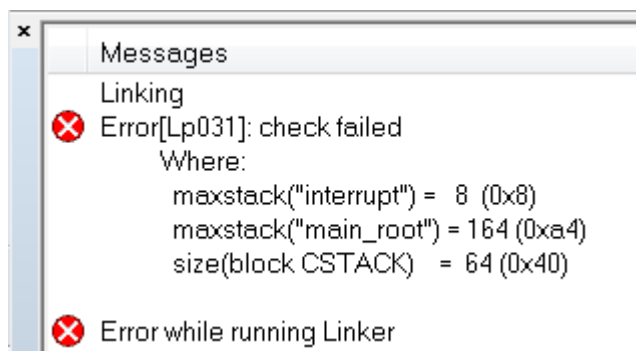
Linker - Stack usage analysis



- 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);
```

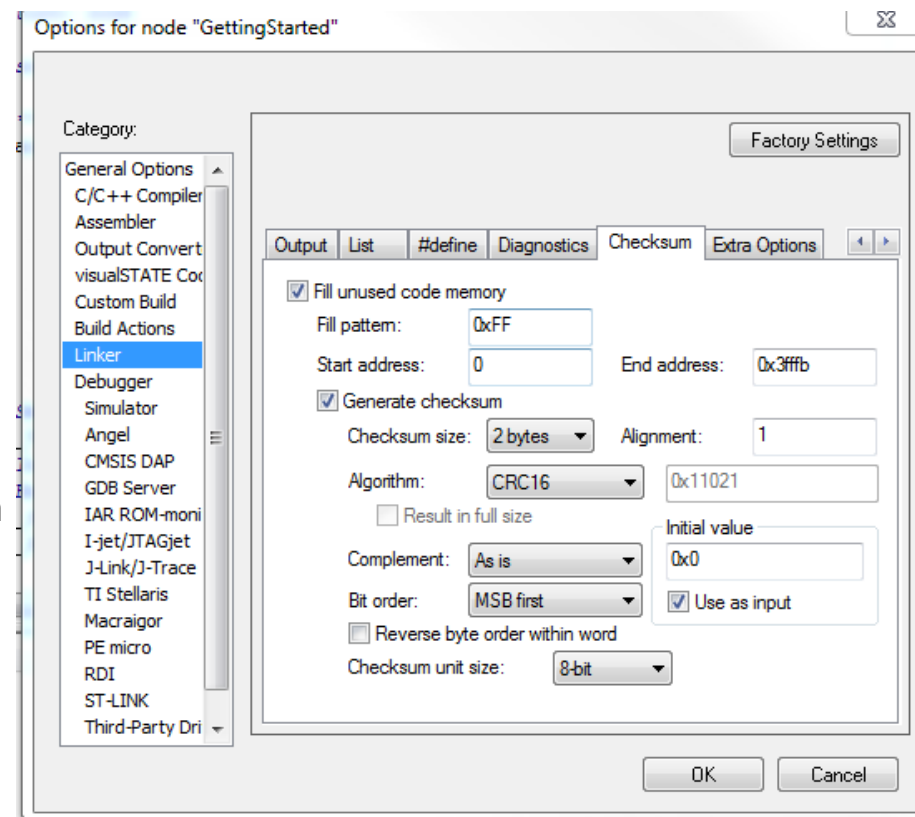
linker script (.icf file)



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>



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

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

The screenshot displays the C-SPY Simulator interface with three main panels:

Function Profiler:

Function	Calls	Flat Time	Flat Time (%)	Acc. Time	Acc. Time (%)
printf(char const*)	17	41460	96.31	41460	96.31
MyRecursiveFunction(int)	17	787	1.83	55478	128.87
main()	2	25	0.06	42272	98.20
delay(long)	1	0	0.00	0	0.00
InitFib()	0	0	0.00	0	0.00
NextCounter()	0	0	0.00	0	0.00
PutFib(unsigned int)	0	0	0.00	0	0.00
GetFib(int)	n	n	n nn	n	n nn

Code Coverage:

- project1 38.18%
- Tutor 60.71%
- DoForegroundProcess() 0.00%
- MyRecursiveFunction(int) 100.00%
- NextCounter() 0.00%
- delay(long) 80.00%
- main() 50.00%
- Utilities 0.00%
- myprintf 100.00%

Interrupt Setup...

- ☒ Interrupt Setup...
- Forced Interrupt
- Interrupt Status Window
- Interrupt Log
- Interrupt Log Summary
- Data Log
- Data Log Summary
- Memory Access Setup...
- Trace
- Function Trace
- Function Profiler
- Timeline
- Breakpoint Usage

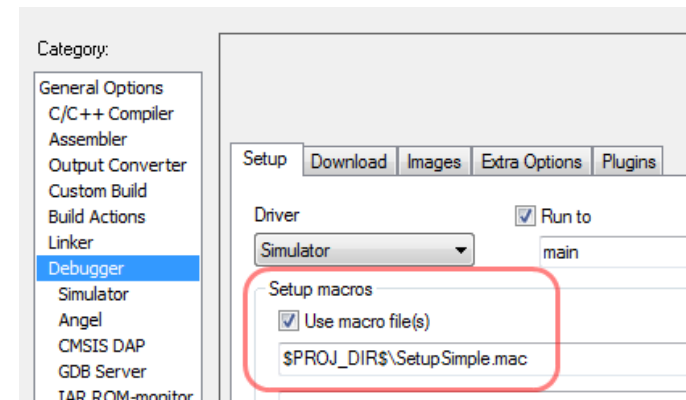
Trace Window:

#	Cycles	Trace
198	376	0x1ea2: CMP R1, R4
199	377	0x1ea4: BNE.N 0x1e98
200	382	0x1ea6: POP {R4, PC}
_call_main:		
201	383	0x1f3c: MOVS R0, #0
202	386	0x1f3e: BL main ...
void main(void)		
{		
main:		
203	388	0x1c18: PUSH {R4, LR}
callCount = 0;		
204	390	0x1c1a: LDR.N R0, ??Da...
205	391	0x1c1c: MOVS R1, #0
206	392	0x1c1e: ...
207	393	0x1c1f: ...



C-SPY macros can be written to help you with complex debugging situations. There are several predefined 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

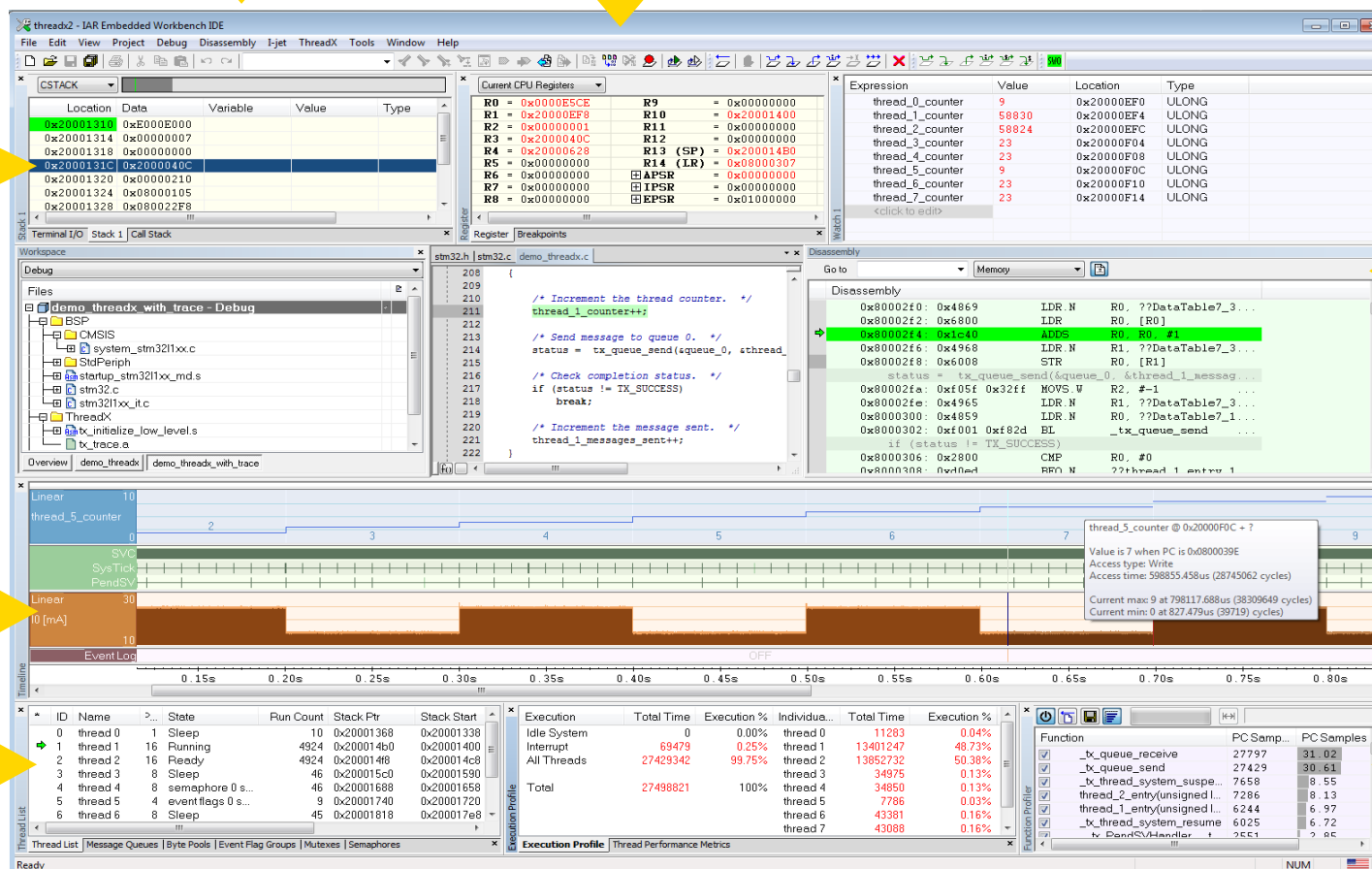
Dockable windows and tab groups

Power visualization

Timeline window

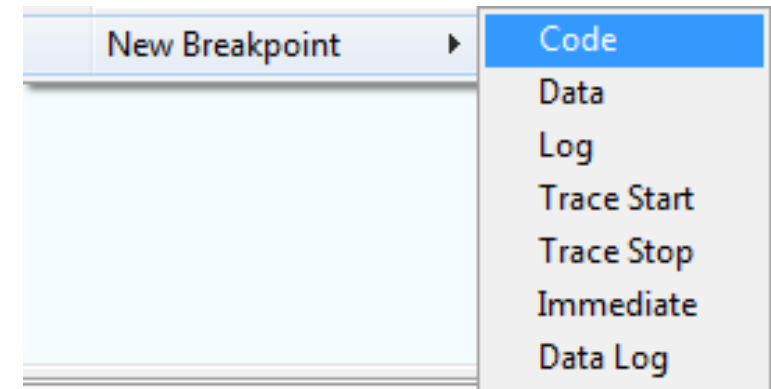
RTOS awareness

Performance analysis



There are more than just “normal” code breakpoints:

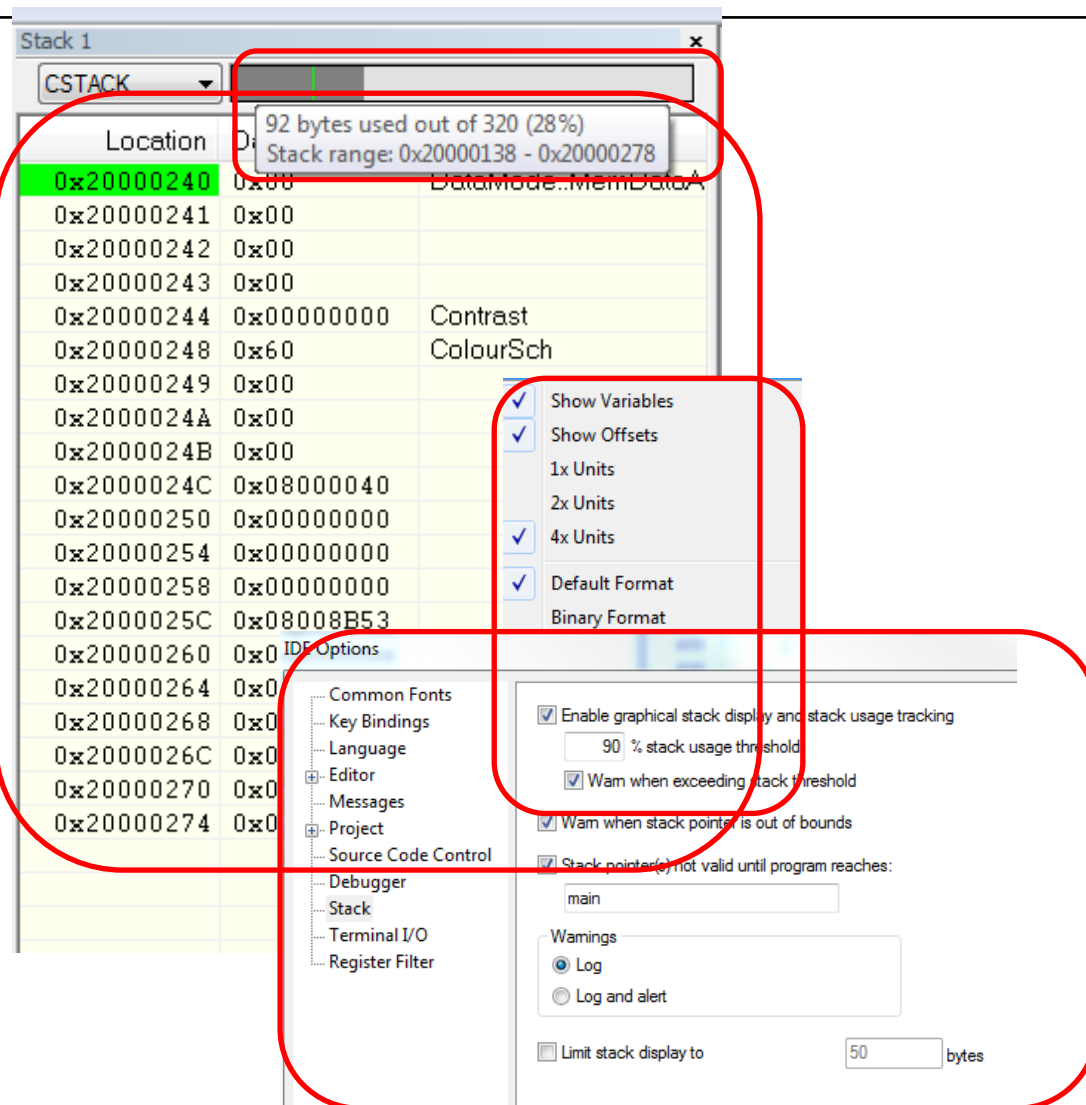
- **Data breakpoint**
halts the target at access of a specific memory location/range set in memory window, optional with a certain pattern match (dependent on driver)
- **Log breakpoint**
logs execution to the debug log window
- **Immediate breakpoint**
simulator only, association with C-SPY macros
-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

Micrium

expresslogic



WITTENSTEIN



RTOS Plugins (Micrium μ C/OS-III)

The screenshot displays the IAR Embedded Workbench IDE interface. The top menu bar includes File, Edit, View, Project, Debug, Disassembly, I-Jet/JTAG, uC/OS-III, Tools, Window, and Help. The main workspace is divided into several panes:

- uC/OS-III Status:** Shows the kernel is running (Version 3.03.01). It includes statistics for CPU usage (n/a), tasks (4), idle counter (5968925), and context switches (17716). A 'Reset Statistics' button is available.
- uC/OS-III Task List:** A table listing tasks with columns for Task Name, Priority, State, Pending On Object, Pending On, CPU Usage, Bar Graph, Context Switches, Stack Pointer, Stack Size, Stack Free, Stack Used, Stack Used %, and another Bar Graph.
- Workspace:** Shows the project structure on the left, including APP, BSP, uC/CPU, uC/LIB, uC/OS-III, and Output. The right pane shows the source code for 'os_core.c'.

The task list table contains the following data:

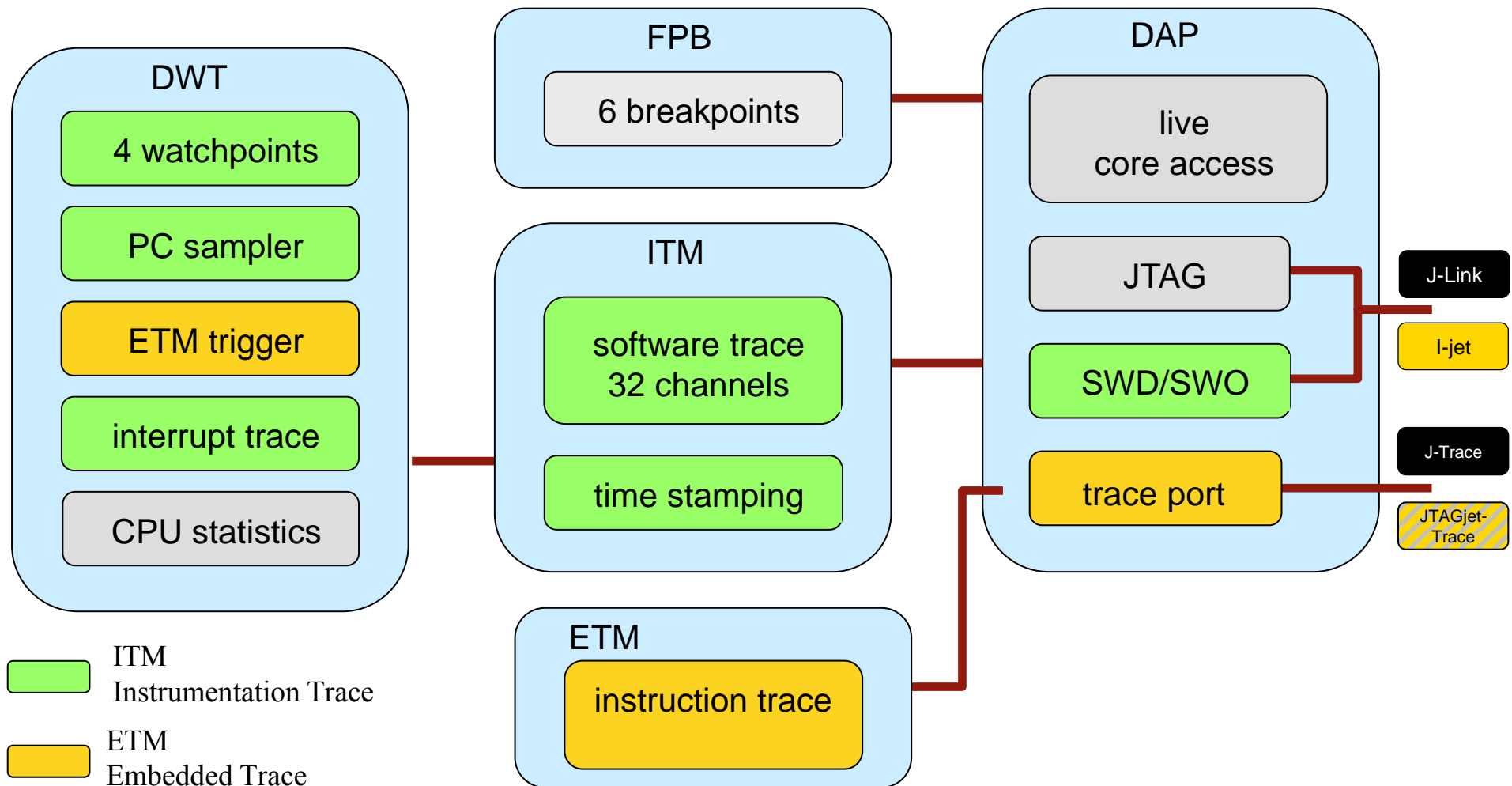
#	Task Name	Priority	State	Pending On Object	Pending On	CPU Usage	Bar Graph	Context Switches	Stack Pointer	Stack Size	Stack Free	Stack Used	Stack Used %	Bar Graph
0	App Task Start	2	Delayed			0.00%		17	0x20000EC0	64	0	0	0%	
1	uC/OS-III Timer Task	8	Pending	Task Semaphore	Task Sem	0.00%		89	0x200010C0	64	0	0	0%	
2	uC/OS-III Tick Task	7	Pending	Task Semaphore	Task Sem	0.00%		8805	0x20000DE8	128	0	0	0%	
3	uC/OS-III Idle Task	9	Ready			0.00%		8805	0x20001010	64	0	0	0%	

The source code in the workspace pane shows the implementation of the CPU wait-for-interrupt function:

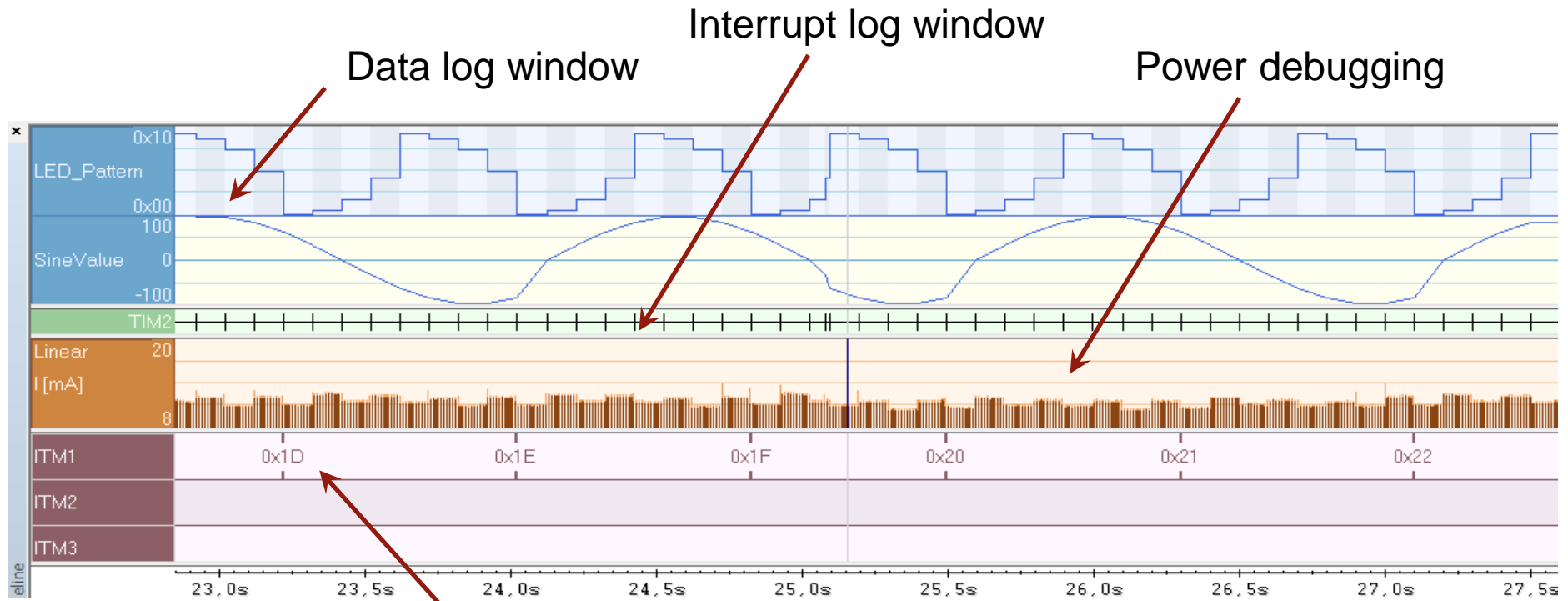
```

96  ;
97  ; CPU_CRITICAL_EXIT(); /* CPU_SR_Restore(cpu_sr); */
98  ;
99  ;
100 ;
101 ;
102 CPU_SR_Save
103     MSR    R0, PRIMASK ; Set prio int mask to mask all (except faults)
104     CPSID  I
105     BX     LR
106
107 CPU_SR_Restore
108     ; See Note #2.
109     MSR    PRIMASK, R0
110     BX     LR
111
112 ;PAGE#1
113 ;
114 ;*****
115 ; WAIT FOR INTERRUPT
116 ;
117 ; Description : Enters sleep state, which will be exited when an interrupt is received.
118 ;
119 ; Prototypes : void CPU_WaitForInt (void)
120 ;
121 ; Argument(s) : none.
122 ;*****
123
124 CPU_WaitForInt
    
```

CoreSight Overview (ARM Cortex-M3/M4)



ITM/SWO Usage (example)



ITM event with line graph

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

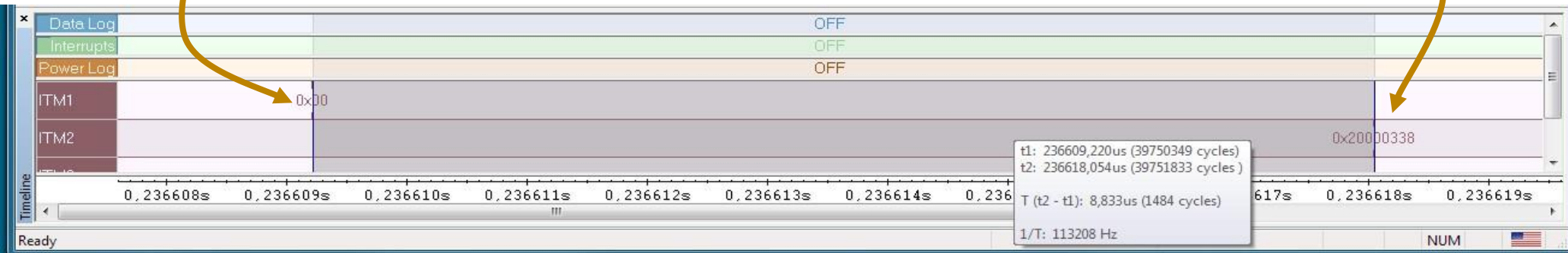
ITM events



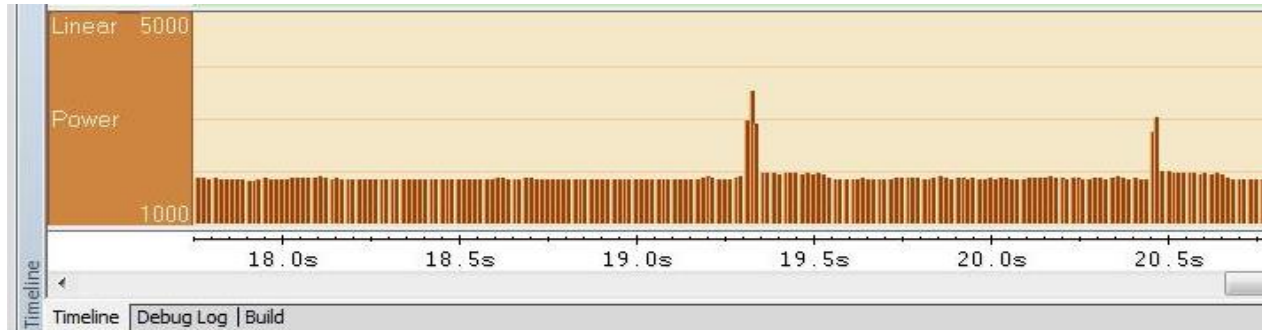
Include the headerfile `arm_itm.h`, here the macros are defined.

- `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());
```



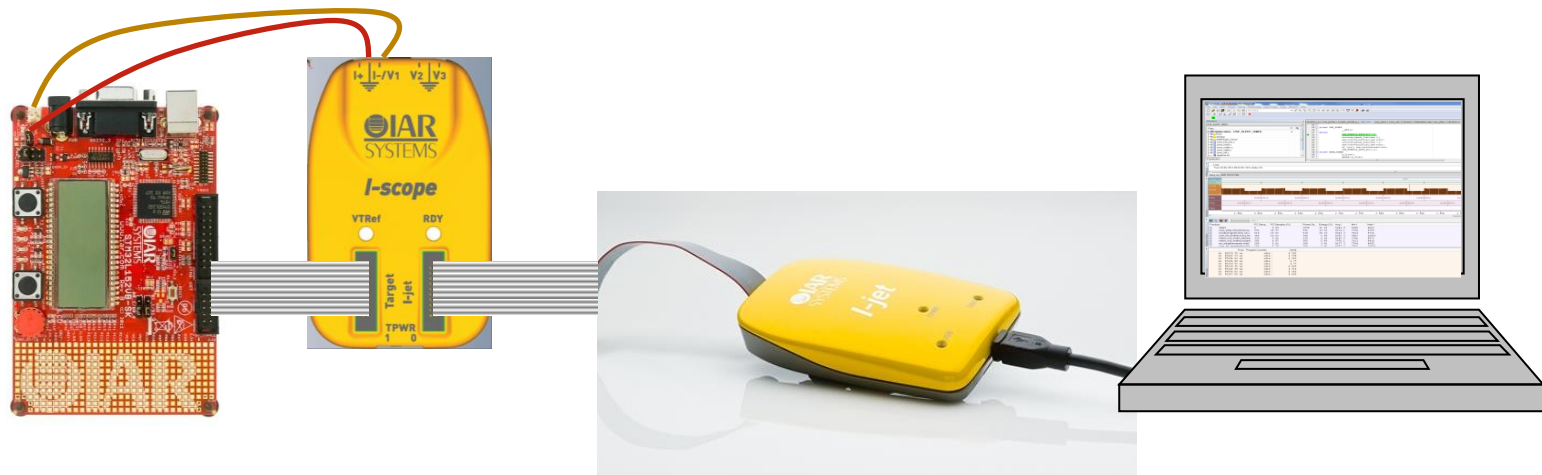
Power Debugging



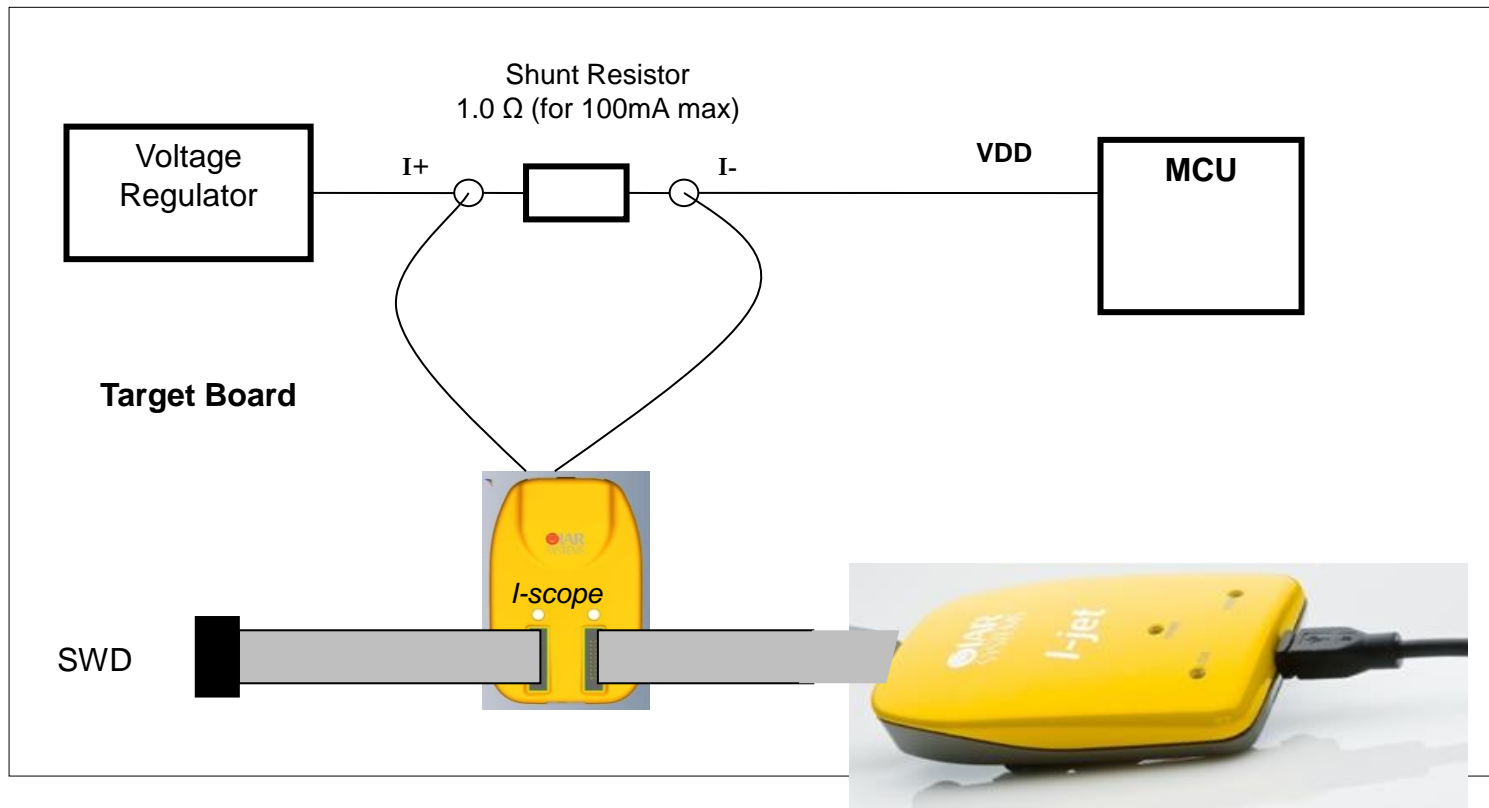
- In-circuit measurement probe from IAR systems
- Used in combination with I-jet
- V1 – V3 voltage range 0 to 6V
- I+ and I- differential voltage, 110mV full scale
- Sampling rate up to 200 kHz, 12 bit resolution



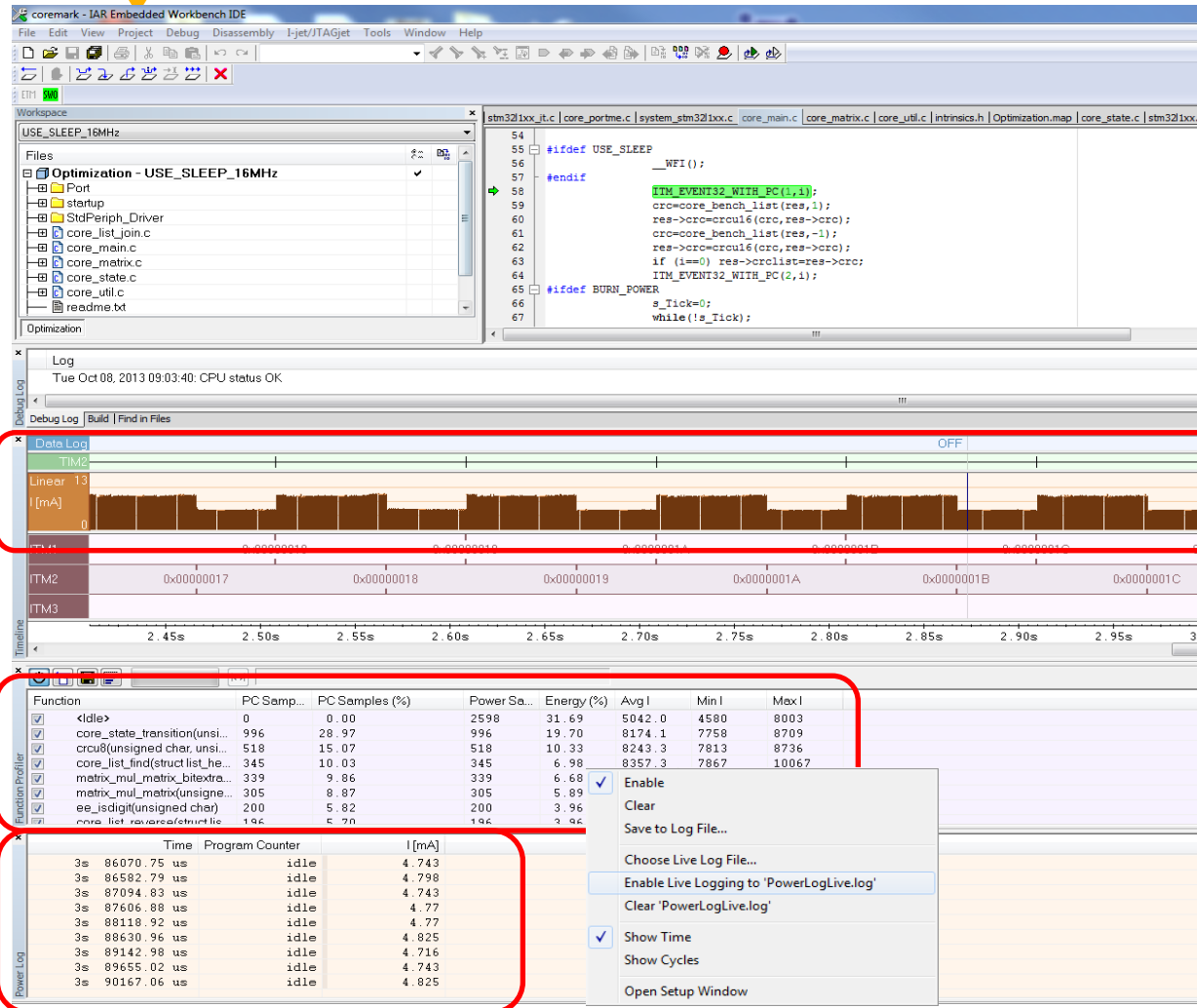
Current-/ voltage measurement



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

Workspace

TRACE

Files

- GettingStar...
- BSP
- CMSIS
- Other
- StdPeriph...
- Vectors
- main.c
- readme.txt
- stm3211x...
- Output

Information Center for ARM | main.c | arm_itm.h | system_stm3211x.c | readme.txt

```
311 /*
312 static void SetSysClock(void)
313 {
314     __IO uint32_t StartUpCounter = 0, HSEStatus = 0;
315
316     /* SYSCLK, HCLK, PCLK2 and PCLK1 configuration -----*/
317     /* Enable HSE */
318     RCC->CR |= ((uint32_t)RCC_CR_HSEON);
319
320     /* Wait till HSE is ready and if Time out is reached exit */
321     do
322     {
323         HSEStatus = RCC->CR & RCC_CR_HSERDY;
324         StartUpCounter++;
325     } while((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));
326
327     if ((RCC->CR & RCC_CR_HSERDY) != RESET)
328     {
329         HSEStatus = (uint32_t)0x01;
330     }
331     else
332     {
333         HSEStatus = (uint32_t)0x00;
334     }
335
336     if (HSEStatus == (uint32_t)0x01)
337     {
338         /* Enable 64-bit access */
339         FLASH->ACR |= FLASH_ACR_ACC64;
340
341         /* Enable Prefetch Buffer */
```

GettingStarted

#	Cycl...	Address	Trace	Exec	Exc...	Access	Data Address	Data Value	Comment
558	-	0x20000570	} while((HSEStatus == 0) && (St...	Thu...					
559	-	0x20000570	} while((HSEStatus == 0) && (St...	Thu...					
560	-	0x20000572	LDR R0, [SP]	Thu...					
561	1180	0x20000576	CMP R0, #0	Thu...					
562	-	0x20000578	LDR R0, [SP, #0x4]	Thu...					
563	-	0x2000057c	CMP W R0, #1280	Thu...					
564	1204	0x20000560	BNE N ??SetSysClock_0	Thu...					
565	-	0x20000562	HSEStatus = RCC->CR & RCC_CR_...	Thu...					
566	-	0x20000564	LDR R0, ??DataTable2	Thu...					
567	-	0x20000568	LDR R0, [R0]	Thu...					
568	-	0x2000056a	ANDS W R0, R0, #131072	Thu...					
569	-	0x2000056c	STR R0, [SP]	Thu...					
570	-	0x2000056e	StartUpCounter++;	Thu...					
571	-	0x20000570	LDR R0, [SP, #0x4]	Thu...					
572	-	0x20000572	ADDS R0, R0, #1	Thu...					
573	-	0x20000574	CTP R0, [SP, #0x4]	Thu...					



ETM Trace features

Instruction
Profiling



Disassembly

Address	Disassembly	Comment
408	0x800295e: 0xe003 B.N	??core_list_revers...
12240	0x8002960: 0x6802 LDR R2, [R0]	
12240	0x8002962: 0x6001 STR R1, [R0]	
12240	0x8002964: 0x0001 MOVS R1, R0	
12240	0x8002966: 0x0010 MOVS R0, R2	

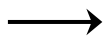
Function
Trace



ETM Function Trace

#	Cycles	Address	Trace	Exec	Exc...	Access	Data
1797339	4008946	0x08003010	core_state_transition(unsig...	Thumb			
1797404	4009048	0x08002f04	ee_isdigit(unsigned char)	Thumb			
1797415	4009062	0x08003044	core_state_transition(unsig...	Thumb			
1797450	4009116	0x08002f04	ee_isdigit(unsigned char)	Thumb			
1797461	4009130	0x08003044	core_state_transition(unsig...	Thumb			
1797496	4009184	0x08002f04	ee_isdigit(unsigned char)	Thumb			
1797507	4009198	0x08003044	core_state_transition(unsig...	Thumb			
1797542	4009256	0x08002f04	ee_isdigit(unsigned char)	Thumb			

ETM Trace



ETM Trace

#	Cycles	Address	Trace	Exec	Exc...	Access	Data Address	Data Value	Comment
541910	-	0x08002966	MOVS R0, R2	Thu...					
541911	-	0x08002968	while (list) {	Thu...					
541912	-	0x0800296a	tmp=list->next;	Thu...					
541913	1353684	0x08002960	LDR R2, [R0]	Thu...					
541914	-	0x08002962	list->next=next;	Thu...					

Function
Profiler



Function Profiler

Function	Calls	Flat Time	Flat Time (%)	Acc. Time	Acc. Time (%)
main(int, char **)	2	4916848	52.78	6002178	64.43
core_state_transition(unsigned char **, unsigned int *)	4096	1117004	11.99	1244306	13.36
<Other>	0	806980	8.66	5063872	54.35
crcu8(unsigned char, unsigned short)	2149	483124	5.19	483124	5.19
matrix_mul_matrix_bitextract(unsigned int *, signed short *, signed short *)	16	371514	3.99	371514	3.99
matrix_mul_matrix(unsigned int *, signed short *, signed short *)	16	336670	3.61	336670	3.61
core_list_find(struct list_head_s *, struct list_data_s *)	721	307014	3.30	307014	3.30

Code
Coverage



Code Coverage

Function	Coverage
core_list_join	74.21%
calc_func(signed short *, struct RESULTS_S *)	91.67%
cmp_complex(struct list_data_s *, struct list_data_s *, struct RESULTS_S *)	100.00%
cmp_idx(struct list_data_s *, struct list_data_s *, struct RESULTS_S *)	100.00%
copy_info(struct list_data_s *, struct list_data_s *)	0.00%
core_bench_list(struct RESULTS_S *, signed short)	100.00%
core_list_find(struct list_head_s *, struct list_data_s *)	100.00%
core_list_init(unsigned int, struct list_head_s *, signed short)	0.00%

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

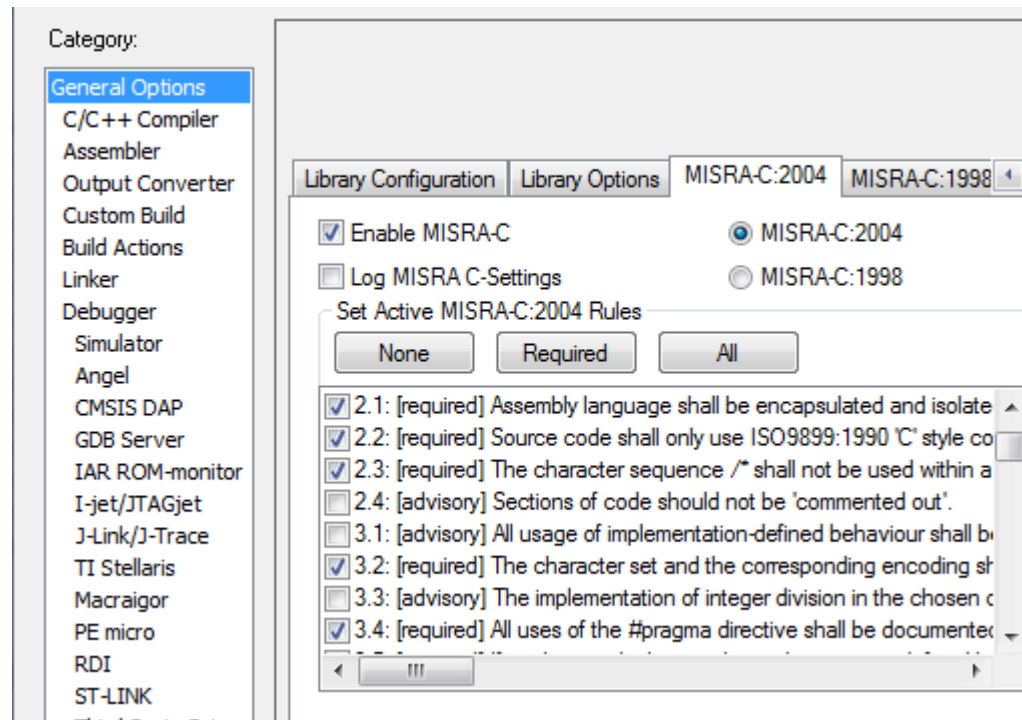


- Overview
- IAR Embedded Workbench IDE vs. Eclipse
- Compiler
- Linker
- Debugger
- Safety
- IAR Academy
- XMC4500 lab and demonstration

Safety – MISRA-C



- MISRA-C:2004 Checker included



Support for MISRA-C:2012 planned for 2014

Skip MISRA



MISRA C 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 reliability
- 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; /* ? */`



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--)  
    {  
        ...  
    }  
}
```



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
```



This rule states:

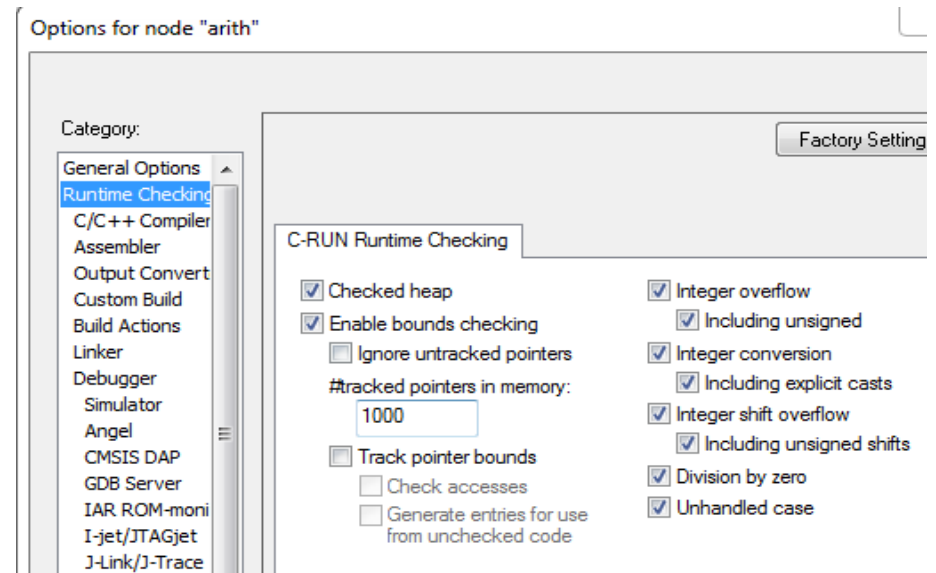
In the definition of a function-like macro, each instance of a parameter shall be enclosed in parenthesis ()

Example:

```
#define TIMES_TWO(x) (x) * 2
```

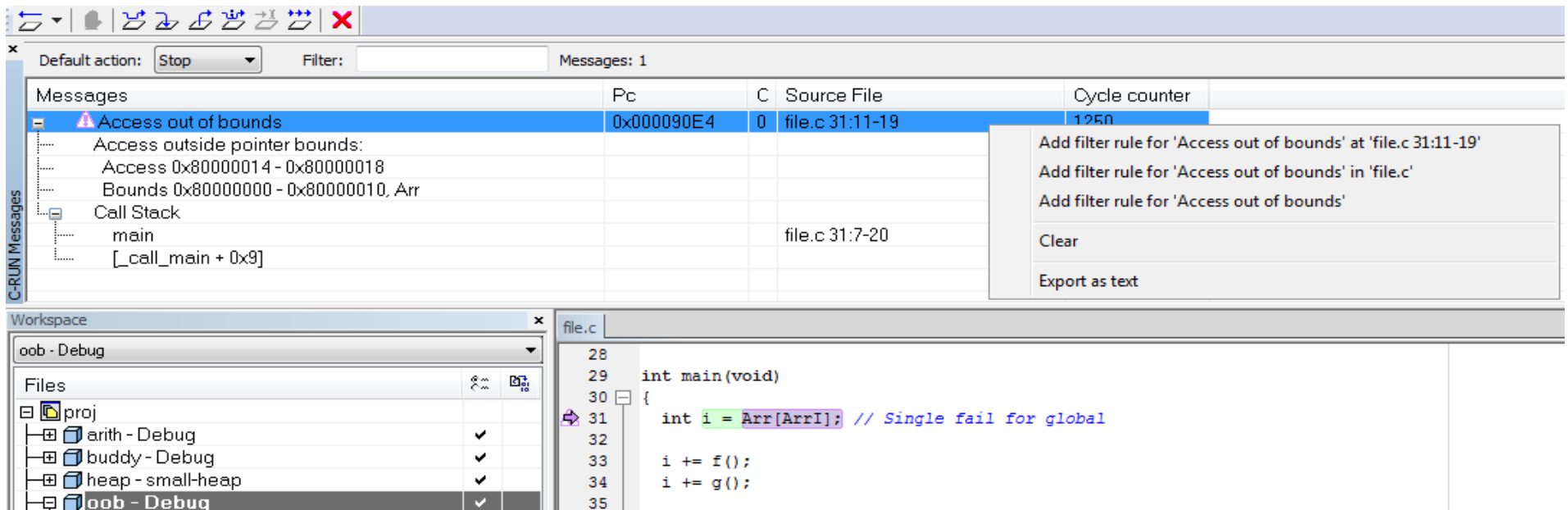
```
a=TIMES_TWO(2+3);    //      (2+3) * 2  =  10
```

- ***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***



The screenshot displays the IAR Embedded Workbench interface during a runtime analysis session. The 'C-RUN Messages' window is open, showing a table of messages. The first message is 'Access out of bounds' at PC 0x000090E4, C 0, Source File 'file.c 31:11-19', and Cycle counter 1250. The message details show 'Access outside pointer bounds: Access 0x80000014 - 0x80000018, Bounds 0x80000000 - 0x80000010, Arr'. The call stack shows 'main' and '[_call_main + 0x9]'. A context menu is open over the message, offering options: 'Add filter rule for 'Access out of bounds' at 'file.c 31:11-19'', 'Add filter rule for 'Access out of bounds' in 'file.c'', 'Add filter rule for 'Access out of bounds'', 'Clear', and 'Export as text'.

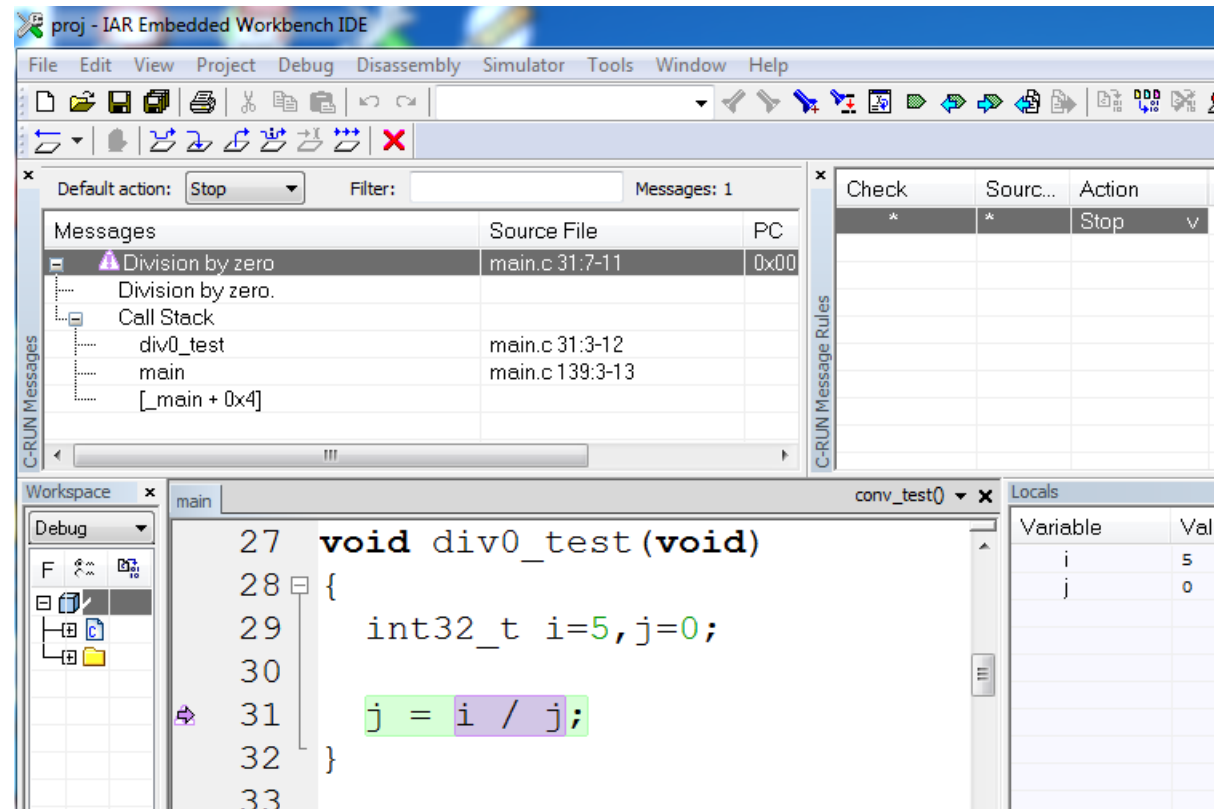
The 'Workspace' window shows a project named 'proj' with several sub-projects: 'arith - Debug', 'buddy - Debug', 'heap - small-heap', and 'oob - Debug' (selected).

The 'file.c' window shows the source code at line 31:11-19:

```
28
29 int main(void)
30 {
31     int i = Arr[ArrI]; // Single fail for global
32
33     i += f();
34     i += g();
35 }
```

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

The screenshot displays the IAR Embedded Workbench IDE interface. The top menu bar includes File, Edit, View, Project, Debug, Disassembly, Simulator, Tools, Window, and Help. Below the menu is a toolbar with various icons. The main workspace shows a C program with the following code:

```
34 void conv_test()  
35 {  
36     int32_t i=5;  
37     uint8_t ch=0x50;  
38  
39     ch+=0x105;  
40     ch = (uint8_t) ( i * 100);  
41 }
```

The C-RUN Messages window is open, showing two error messages:

- Division by zero (Source File: main.c 31:7-11)
- Integer conversion failure (Source File: main.c 39:3-12)

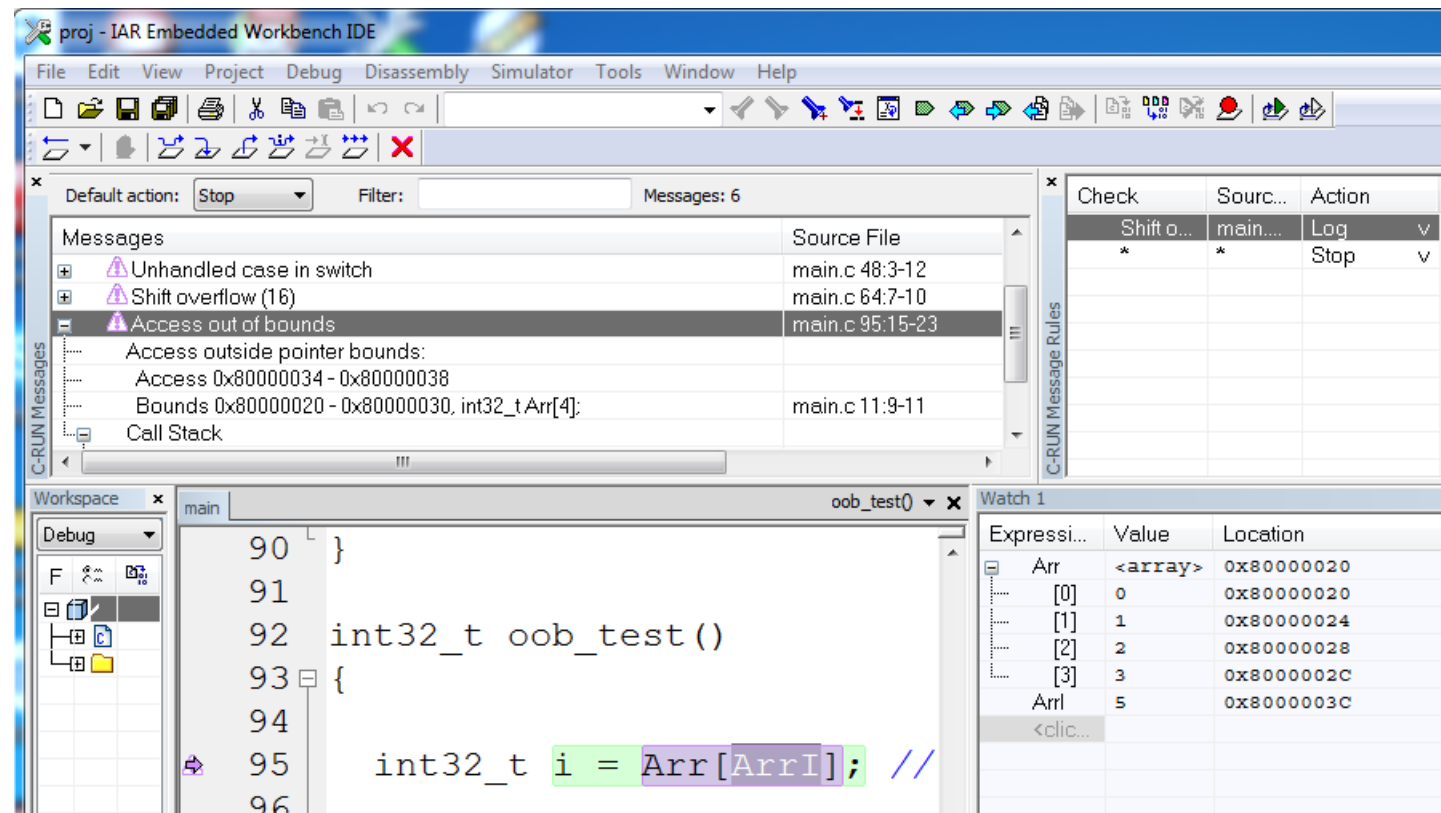
The Integer conversion failure message is expanded, showing the conversion changes the value from 341 (0x000000155) to 85 (0x55). The Call Stack shows the function conv_test in main.c 39:3-12, called from main.c 140:3-13.

The Locals window shows the following variables and values:

Variable	Value
i	5
ch	0x50

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

1	Introduction	2
1.1	Revision history	2
2	Compiler package.....	2
3	Compiler target platforms.....	2
4	Domain of use	2
5	IAR Quality management system (QMS)	2
6	Acceptance testing.....	3
6.1	Test suites	3
6.2	Test engine	5
6.3	Testing during development.....	5
6.4	Product release acceptance.....	5
7	Handling of field reported issues	5
8	End user product documentation	6
9	Product update and maintenance history	6

EWARM – TÜV certified tool chain




IAR
SYSTEMS

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)



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

ZERTIFIKAT ♦ **CERTIFICATE** ♦ **CERTIFICADO** ♦ **CERTIFIKAT** ♦ **СЕРТИФИКАТ** ♦ **ISO 9001** ♦ **ISO 14001** ♦ **ISO 26262** ♦ **ISO 27001** ♦ **ISO 29001** ♦ **ISO 31000** ♦ **ISO 45001** ♦ **ISO 50001** ♦ **ISO 9001** ♦ **ISO 14001** ♦ **ISO 26262** ♦ **ISO 27001** ♦ **ISO 29001** ♦ **ISO 31000** ♦ **ISO 45001** ♦ **ISO 50001**

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

Dissemination, distribution, copying or any other use of any information in this report in part is strictly prohibited.

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

- 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

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

Exercises and Demo