# Take full control of your MCU software development

*Ryan Sheng*
*ryan.sheng@iar.com*

IAR SYSTEMS

# IAR Systems

168 Employees with HQ in Uppsala, Sweden

Listed on Stockholm/NASDAQ

R&D investment 32% of revenue

34 years in the embedded industry

Uppsala
Munich
Paris
Tokyo
Shanghai
Seoul
San Francisco
Los Angeles
Dallas
Boston

**Global technical support in**

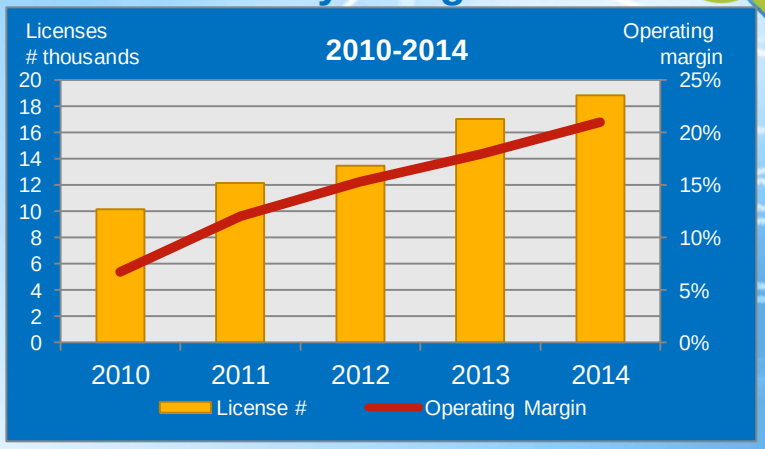
**9 languages**



**Distributor representation in**

**43 countries**

## Stability and growth




# IAR Embedded Workbench
## *C/C++ compiler and debugger toolchain*

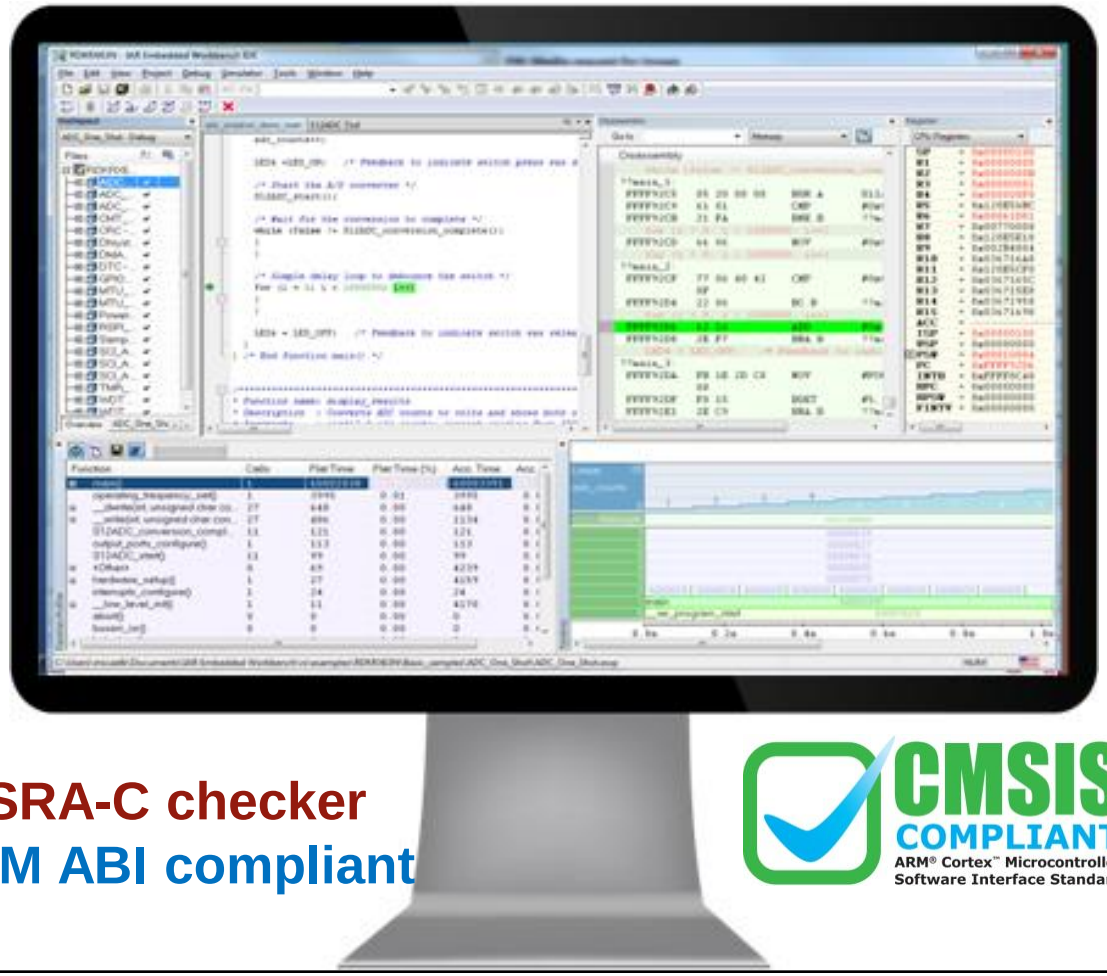
IAR SYSTEMS

**Outstanding optimization for both compact code size and high performance**

**Comprehensive debugger**

**User-friendly features and broad ecosystem integration**

**Global technical support**



Safety tested · Production monitored · TÜV SÜD · Functional Safety

**MISRA-C checker**

**ARM ABI compliant**

CMSIS COMPLIANT
ARM® Cortex™ Microcontroller Software Interface Standard

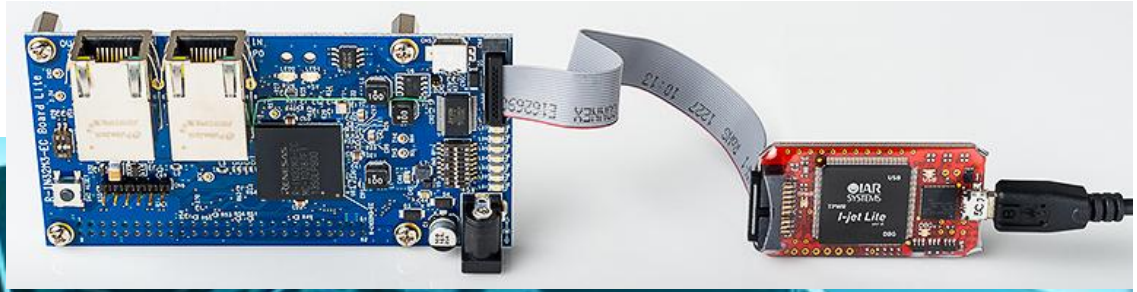
# IAR Embedded Workbench
## *Device support for NXP MCU/MPU*

- **IAR Embedded Workbench for ARM**
  - LPC, Kinetis
  - i.MX, Vybrid
  - S32K, MAC57D5xx
- **IAR Embedded Workbench for ColdFire**
  - V1, V2, V3, ColdFire+
- **IAR Embedded Workbench for HCS12**
  - 16-bit HC12, S12 devices
- **IAR Embedded Workbench for S08**
  - 8-bit S08 devices
- **IAR Embedded Workbench for 8051**
  - 8-bit LPC7xx, LPC9xx, …

# IAR Embedded Workbench for ARM

- Unique independence with support for all available ARM cores, from all major vendors including NXP, ST, TI, Cypress, Microchip, Renesas, SiLabs, Toshiba, etc.
- 5,000+ supported devices
- Close cooperation with SoC vendors

**IAR KickStart Kit**





CMSIS COMPLIANT
ARM® Cortex™ Microcontroller Software Interface Standard

ARM

Cortex-A17
Cortex-A15
Cortex-A9
Cortex-A8
Cortex-A7
Cortex-A5
Cortex-R8
Cortex-R7
Cortex-R5
Cortex-R4
Cortex-M7
Cortex-M4
Cortex-M3
Cortex-M1
Cortex-M33
Cortex-M23
Cortex-M0/M0+
ARM10/11
ARM7/9

# IAR Embedded Workbench for ARM
## *Recent updates and highlights*

- **Updated IDE look and feel**
- **Support C11 (ISO/IEC 9899:2011) standard**
- **Support C++14 (ISO/IEC 14882:2014) standard**
- **Enhanced compiler optimizations (for speed)**
- **Support Cortex-M23 & Cortex-M33 (ARMv8-M)**
- **CMSIS-Pack**
- **I-jet Trace for ARM (Cortex-A/R/M)**
- **Flash breakpoint**
- **Multi-core debugging**
- **Stack usage analysis**
- **Enhanced static and runtime code analysis**

# IAR Embedded Workbench for ARM
## *Outstanding compiler optimization*

IAR SYSTEMS

- Leading compiler optimizations enable IAR Embedded Workbench to generate the most compact and fast performing code.
  - **Smaller memory size**
  - **Better real-time reaction**
  - **Lower power consumption**


- **EEMBC** proved code performance: Find the leading **CoreMark®** scores on the website.

www.eembc.org/coremark

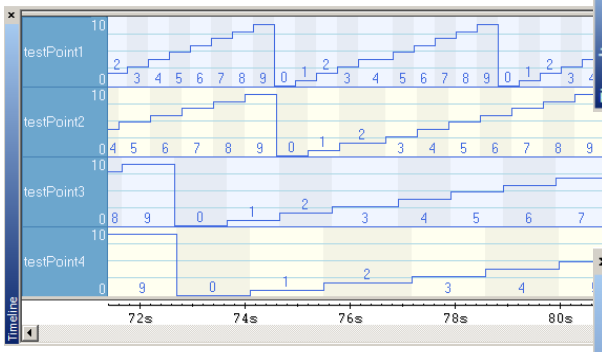
| | Processor | Compiler | Operating Speed in Mhz | CoreMark /MHz (1) | CoreMark (1) |
|---|---|---|---|---|---|
| Cortex-M7 | NXP Kinetis KV5x | IAR v7.50.3 | 240 | 5.00 | 1211.00 |
| Cortex-M4 | NXP Kinetis K70 90nm | IAR v6.50 | 150 | 3.40 | 510.02 |
| Cortex-M0+ | NXP KL25 | IAR 6.60 | 48 | 2.41 | 115.46 |
| Cortex-M0+ | NXP KL05 | IAR 6.60 | 48 | 2.42 | 116.18 |

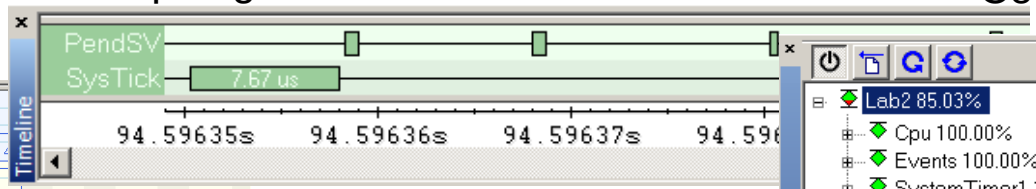
# IAR Embedded Workbench for ARM
## *Advanced debugging & trace*

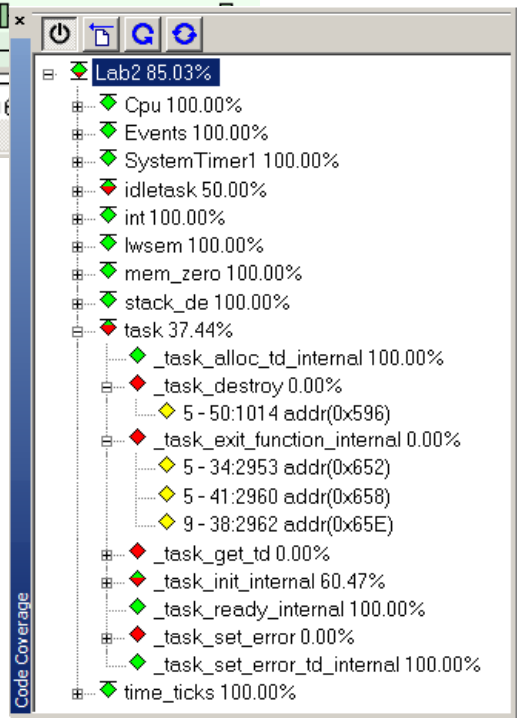
Interrupt log

Code coverage

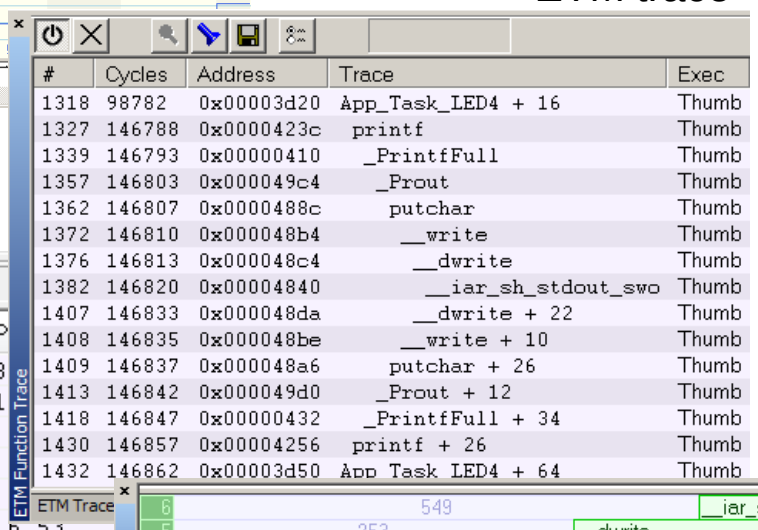
Data log



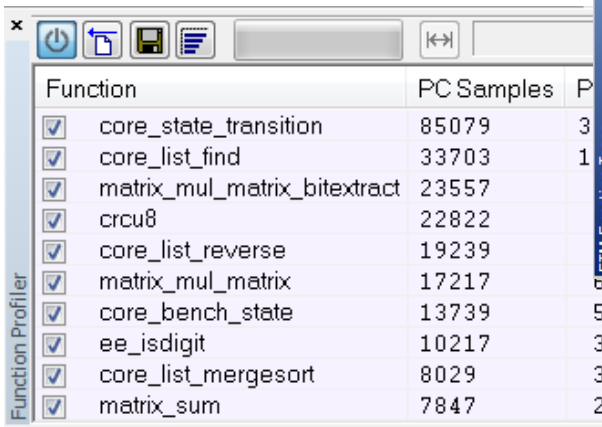

ETM trace



| # | Cycles | Address | Trace | Exec |
|---|---|---|---|---|
| 1318 | 98782 | 0x00003d20 | App_Task_LED4 + 16 | Thumb |
| 1327 | 146788 | 0x0000423c | printf | Thumb |
| 1339 | 146793 | 0x00000410 | _PrintfFull | Thumb |
| 1357 | 146803 | 0x000049c4 | _Prout | Thumb |
| 1362 | 146807 | 0x0000488c | putchar | Thumb |
| 1372 | 146810 | 0x000048b4 | __write | Thumb |
| 1376 | 146813 | 0x000048c4 | __dwrite | Thumb |
| 1382 | 146820 | 0x00004840 | __iar_sh_stdout_swo | Thumb |
| 1407 | 146833 | 0x000048da | __dwrite + 22 | Thumb |
| 1408 | 146835 | 0x000048be | __write + 10 | Thumb |
| 1409 | 146837 | 0x000048a6 | putchar + 26 | Thumb |
| 1413 | 146842 | 0x000049d0 | _Prout + 12 | Thumb |
| 1418 | 146847 | 0x00000432 | _PrintfFull + 34 | Thumb |
| 1430 | 146857 | 0x00004256 | printf + 26 | Thumb |
| 1432 | 146862 | 0x00003d50 | App_Task_LED4 + 64 | Thumb |

ETM Function Trace

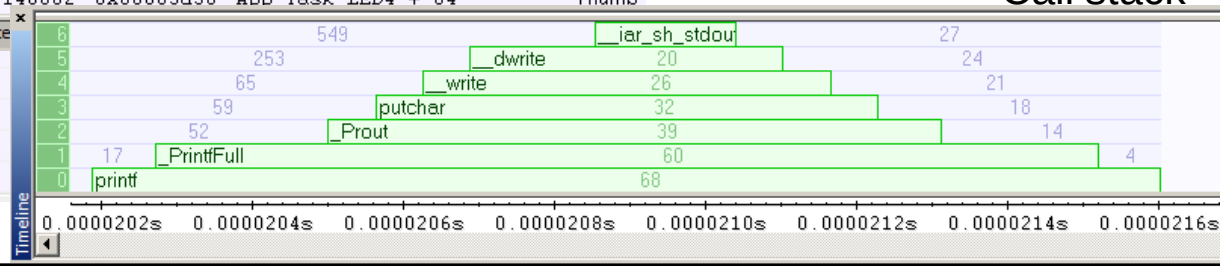
Function profiling



| Function | PC Samples | P |
|---|---|---|
| core_state_transition | 85079 | 3 |
| core_list_find | 33703 | 1 |
| matrix_mul_matrix_bitextract | 23557 | |
| crcu8 | 22822 | |
| core_list_reverse | 19239 | |
| matrix_mul_matrix | 17217 | 6.53 |
| core_bench_state | 13739 | 5.21 |
| ee_isdigit | 10217 | 3.88 |
| core_list_mergesort | 8029 | 3.05 |
| matrix_sum | 7847 | 2.98 |

Function Profiler

Call stack



# IAR Embedded Workbench for ARM
# *Functional safety certification*

IAR SYSTEMS

Functional safety certified edition of
**IAR Embedded Workbench for ARM**

- IEC 61508
- ISO 26262
- EN 50128


## Simplified validation

- Functional safety certificate
- Report of the certificate
- Safety Guide

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 Developme

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. The build tool chain is qualified to be used in safety-related software development according to IEC 6 and ISO 26262. The report IU84911C is a mandatory part of this certifica

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

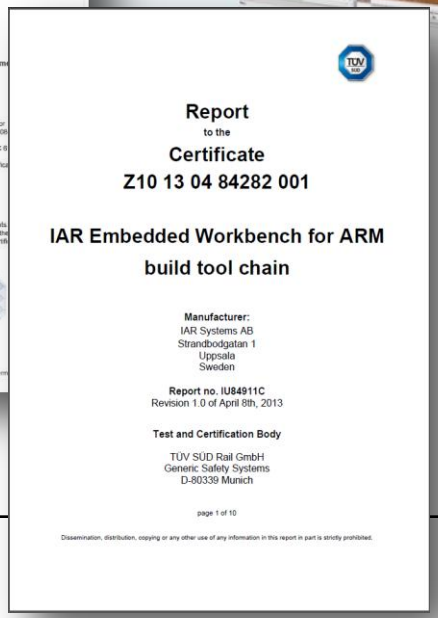
The product was tested on a voluntary basis and complies with the essential requirements certification mark shown above can be affixed on the product. It is not permitted to alter the certification mark in any way. In addition the certification holder must not transfer the certifi to third parties. See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

Test report no.: IU84911C

Date, 2013-04-04 ( Guido Neumann )
Page 1 of 1

472452

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



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.

## Guaranteed support through the product life cycle

- Prioritized technical support
- Validated service packs
- Regular report of known problems

# IAR I-jet & I-jet Trace
## *Hardware debuggers for ARM*

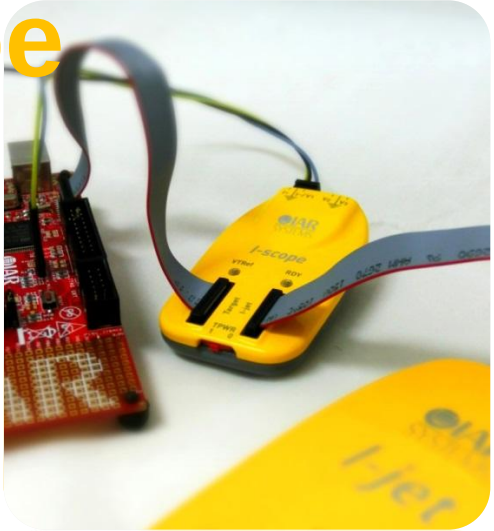
IAR SYSTEMS

## I-jet for ARM

I-jet provides an exceptionally **fast debugging** platform.

## I-scope for ARM

I-scope is a small probe that adds **current & voltage measurement capabilities** to I-jet.



## I-jet Trace for ARM

I-jet Trace delivers **large trace memory capacities** and **high-speed communication** via SuperSpeed USB 3.0.

3rd-Party Debuggers

CMSIS-DAP
Segger J-Link / J-Trace
P&E Micro Multilink
P&E Micro Cyclone
.....

# Generate IAR example projects using NXP MCUXpresso Tools

IAR SYSTEMS

# MCUXpresso Software and Tools

## MCUXpresso Software and Tools

for Kinetis and LPC microcontrollers

### MCUXpresso IDE

Edit, compile, debug and optimize in an intuitive and powerful IDE

### MCUXpresso SDK

Runtime software including peripheral drivers, middleware, RTOS, demos and more

### MCUXpresso Config Tools

Online and desktop tool suite for system configuration and optimization

# MCUXpresso Config Tools

- mcuxpresso.nxp.com/zh/welcome

## MCUXpresso 配置工具

MCUXpresso配置工具提供一套系统配置工具，通过基于Kinetis或LPC的MCU解决方案为各级用户提供帮助。它可引导帮助您完成从初次评估到生产开发的整个过程。


**配置设置**
为您的配置指定可选中间件和环境设置。

**SDK生成器**
生成可下载的SDK存档以与桌面MCUXpresso工具搭配使用

**工程克隆器**
下载一个现存的独立 SDK 示例工程

**引脚工具**
将信号分配给引脚、设置电气属性并生成初始化代码。

**时钟工具**
设置系统时钟并生成初始化代码。

# Create a new configuration

- mcuxpresso.nxp.com/zh/builder
- Select from:
  - Boards
  - Processors
  - Kits
- Select configuration
- Specify configuration settings
- Jump start you configuration

# Configuration settings

# Download a SDK archive

## 下载SDK 生成存档项

如果10秒后未开始下载，请单击以下链接，开始下载软件包:

SDK_2.2_LPCXpresso54114.zip

返回SDK生成器

## SDK生成器

生成可下载的SDK存档以与桌面MCUXpresso工具搭配使用。

**当前配置**

LPCXpresso54114

**查看 SDK 详情**

您的 SDK 下载将包含侧边面板列出的项目。
可使用"工具"->"配置设置"页编辑这些选 配置设置 页

**此MCUXpresso SDK配置可直接下载**

立即下载

软件包名称

SDK_2.2_LPCXpresso54114

# Download an existing example project

## 示例工程

下载一个现有的独立 SDK示例工程或加载某示例中的引脚和时钟数据到在线配置工具。

选择一个示例工程

搜索...

- LPCXpresso54114
  - cmsis_driver_examples
  - rtos_examples
  - driver_examples
  - multicore_examples
  - usb_examples
  - demo_apps
    - hello_world
    - utick_wakeup
    - power_manager_lpc

下载示例 | 应用工程到配置工具

## 下载SDK 生成存档项

如果10秒后未开始下载，请单击以下链接，开始下载软件包:

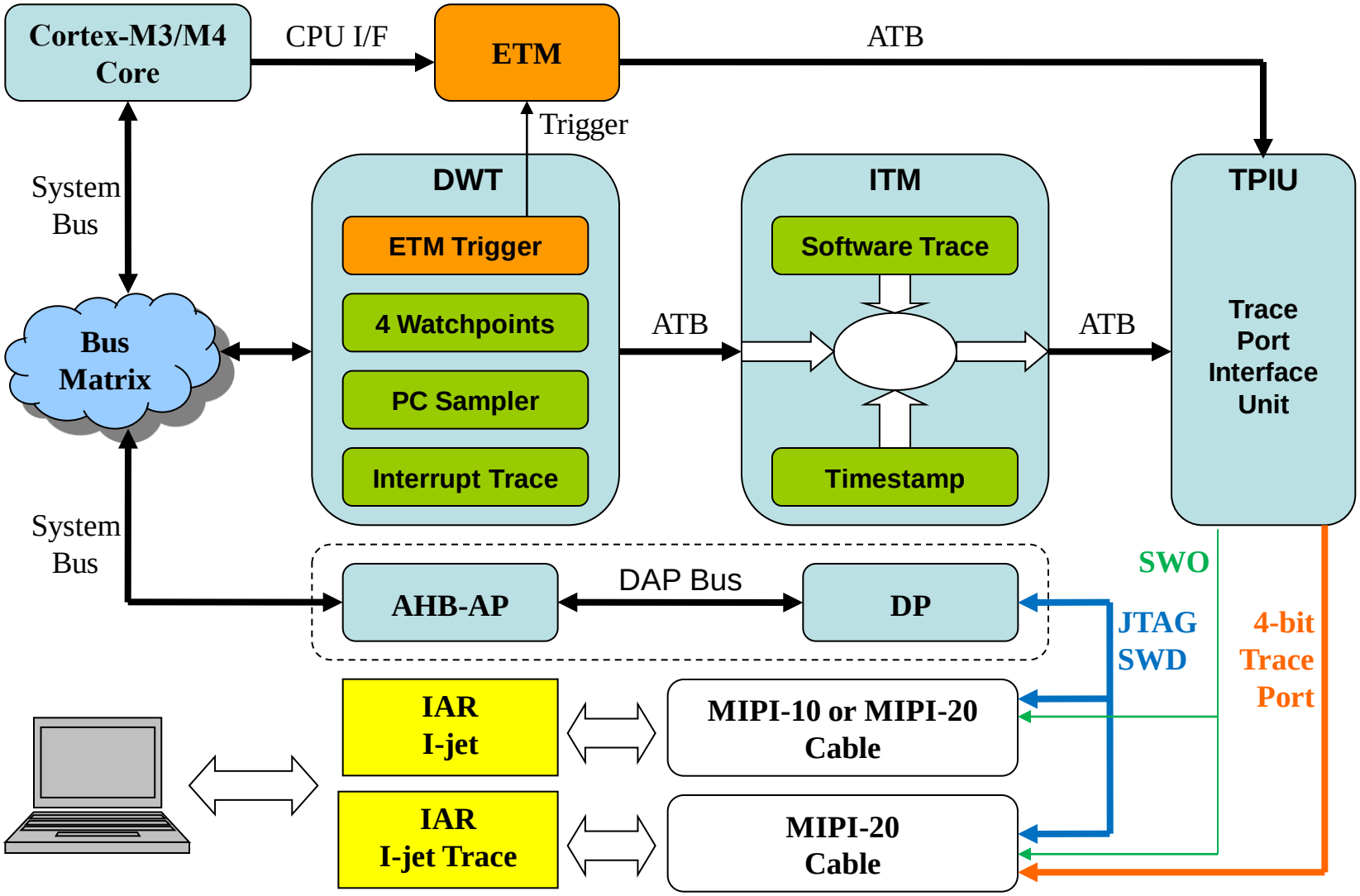
hello_world_cm4.zip

返回示例工程

# Advanced debugging & trace on LPC & Kinetis Microcontrollers

IAR SYSTEMS

# Cortex-M3/M4 trace system



# Trace pins in different connector

## Standard 20-pin 0.1”

| | | | | |
|---|---|---|---|---|
| VTRef | | 1 | 2 | Vtarget |
| NC | (nTRST) | 3 | 4 | GND |
| NC | (TDI) | 5 | 6 | GND |
| SWDIO | (TMS) | 7 | 8 | GND |
| SWCLK | (TCK) | 9 | 10 | GND |
| NC | (RTCK) | 11 | 12 | GND |
| SWO | (TDO) | 13 | 14 | GND |
| nSRST | | 15 | 16 | GND |
| | | 17 | 18 | GND |
| +5V | | 19 | 20 | GND |

JTAG/SWD


## MIPI-10 0.05”

| | | | | |
|---|---|---|---|---|
| VTref | | 1 | 2 | SWDIO / TMS |
| GND | | 3 | 4 | SWCLK / TCK |
| GND | | 5 | 6 | SWO / TDO |
| KEY | | 7 | 8 | NC/EXTb / TDI |
| GNDDetect | | 9 | 10 | nRESET |

JTAG/SWD

## MIPI-20 0.05”

| | | | |
|---|---|---|---|
| VTref | 1 | 2 | SWDIO / TMS |
| GND | 3 | 4 | SWCLK / TCK |
| GND | 5 | 6 | SWO/EXTa/TRACECTL / TDO |
| KEY | 7 | 8 | NC/EXTb / TDI |
| GNDDetect | 9 | 10 | nRESET |
| GND/TgtPwr+Cap | 11 | 12 | TRACECLK |
| GND/TgtPwr+Cap | 13 | 14 | TRACEDATA[0] |
| GND | 15 | 16 | TRACEDATA[1] |
| GND | 17 | 18 | TRACEDATA[2] |
| GND | 19 | 20 | TRACEDATA[3] |

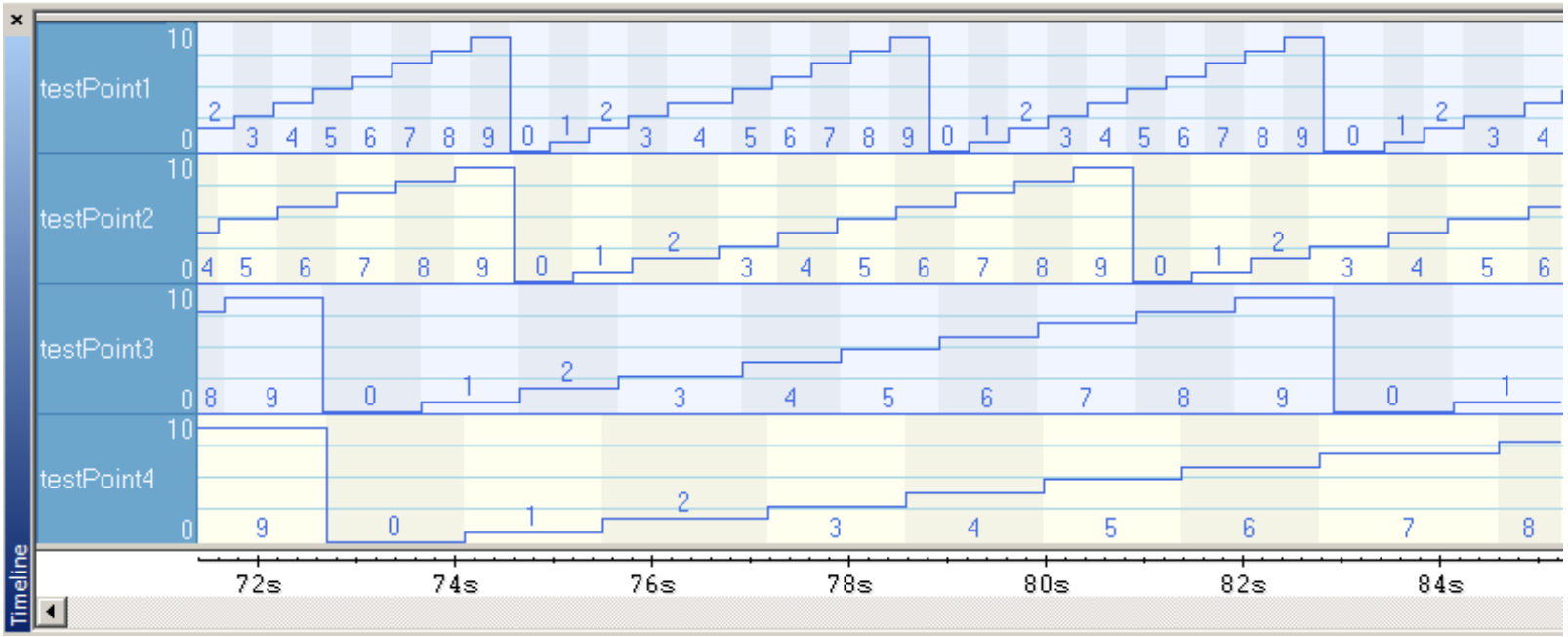
JTAG/SWD/ETM

# SWO trace

- SWO (Serial Wire Output)
  - A serial high speed signal that transmits ITM packets
  - Events and sampling based
  - Supported by Cortex-M3/M4 architectures
  - Supported by IAR I-jet and other debuggers
- Trace information going through SWO
  - DWT (Data Watchpoint and Trace)
    - Watchpoint: 4 independent comparators for address and data
    - PC sampler: Sampling the PC register at regular intervals
    - Interrupt trace: Logging the enter and exit of each interrupt
  - ITM (Instrumentation Trace Macrocell)
    - Generate ITM events on 32 independent ports
    - Packetize and timestamp the DWT events

# Static variables monitoring

- DWT generates trace events when one of the watchpoint finds a match on specified address/data.
- Up to four different static variables can be monitored together.
- C-SPY displays the collected trace information in the Data Log window and the graphical Timeline window.



# Using data log breakpoints

**Right-click on the name of a variable to be monitored:**

- Find in Trace
- Toggle Breakpoint (Code)
- Toggle Breakpoint (Log)
- Toggle Breakpoint (Trace Start)
- Toggle Breakpoint (Trace Stop)
- Toggle Breakpoint (Trace Filter)
- Enable/disable Breakpoint
- Set Data Breakpoint for 'testPoint1'
- **Set Data Log Breakpoint for 'testPoint1'**
- Set Trace Start Breakpoint for 'testPoint1'
- Set Trace Stop Breakpoint for 'testPoint1'
- Set Next Statement
- Quick Watch: 'testPoint1'
- Add to Watch: 'testPoint1'

**Edit Breakpoint**

Data Log

Trigger at: testPoint1 [Edit ...]

Access type: (•) Read/Write ( ) Read ( ) Write

Size: (•) Auto (4) ( ) Manual 4

Trigger range
Requested: 0x20002DE0 - 0x20002DE3
Effective: 0x20002DE0 - 0x20002DE3
[ ] Extend to cover requested range

OK Cancel

**The breakpoint is triggered when the variable at 0x20002DE0 is read or written as a word (4 bytes). DWT will generate an event but the execution will not be stopped.**

- Go to Source
- **Edit...**
- Delete
- Disable
- Enable All
- Disable All
- Delete All
- New Breakpoint ▸

**Check the status and edit the properties of the breakpoint in View → Breakpoints:**

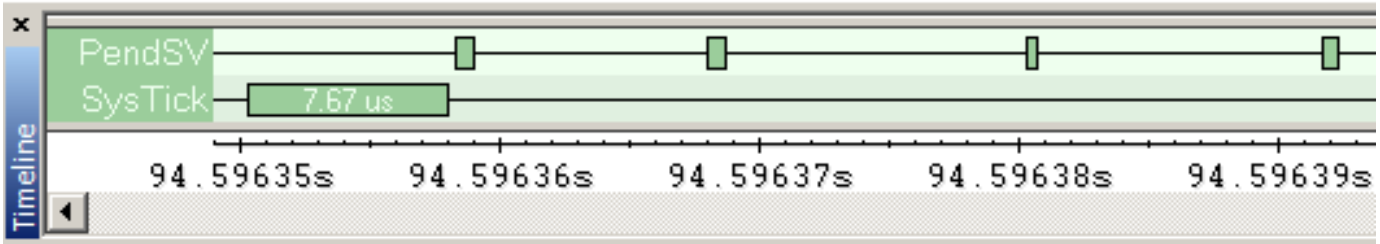
Breakpoints

| | Breakpoint |
|---|---|
| ☑ | Data Log @ testPoint1 [size 4] [Read/Write] [0x20002DE0 – 0x20002DE3] |

variable name — access type — variable address

# Interrupt logging

- DWT generates trace events when entering or leaving any interrupt.
- C-SPY displays the collected trace information in the Interrupt Log window and the graphical Timeline window.
- Useful to find interrupts which can be fine-tuned to execute faster and analyze problems with nested interrupts.




Interrupt Log

| Cycles | Interrupt | Status | Program Counter | Execution Cycles |
|---|---|---|---|---|
| 164998943 | PendSV | Enter | --- | |
| 164998965 | PendSV | Leave | --- | 22 |
| 164999263 | PendSV | Enter | --- | |
| 164999285 | PendSV | Leave | --- | 22 |
| 165045163 | SysTick | Enter | --- | |
| 165045469 | SysTick | Leave | --- | 306 |
| 165093175 | SysTick | Enter | --- | |
| 165093480 | SysTick | Leave | --- | 305 |

# Function profiling

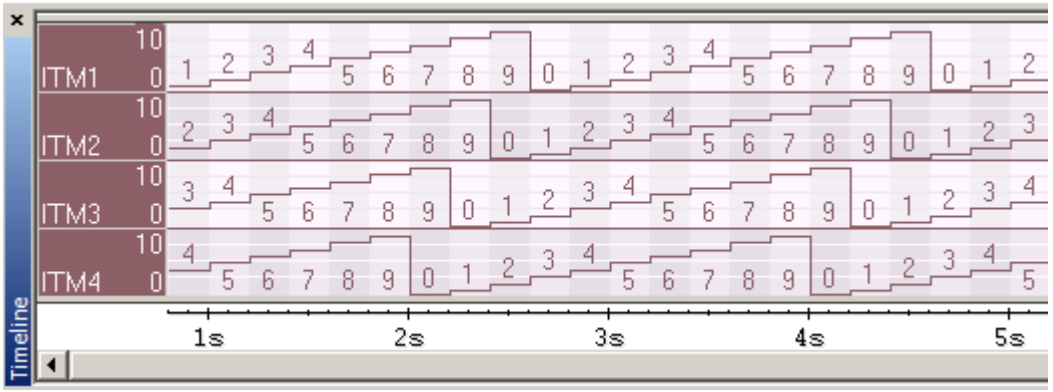
- Code profiling information of each C function is retrieved by counting the number of PC samples generated by the PC sampler of DWT.
- Useful to find where the CPU is spending its time.
- Functions where the most time is spent should be carefully optimized or moved to more efficient memory to increase the performance.

Function Profiler

| Function | PC Samples | PC Samples (%) |
|---|---|---|
| OS_TaskIdle | 1132423 | 46.58 |
| OSTaskIdleHook | 703766 | 28.95 |
| OS_CPU_SR_Save | 288266 | 11.86 |
| OS_CPU_SR_Restore | 196211 | 8.07 |
| App_TaskIdleHook | 93401 | 3.84 |
| OSTimeTick | 11188 | 0.46 |
| OSTaskStkChk | 1987 | 0.08 |
| OS_TaskStatStkChk | 931 | 0.04 |

# Direct output via ITM stimulus ports

- The target application can send data directly to the host debugger through ITM stimulus ports.
- Each of the 32 ITM ports has its own address (based at 0xE0000000).
- C-SPY displays the data sent from ITM port #1~#4 in the Event Log window and the graphical Timeline window.




Event Log

| Cycles | Program Counter | ITM1 | ITM2 | ITM3 | ITM4 |
|---|---|---|---|---|---|
| 438832330 | --- | 2 | | | |
| 438832367 | --- | | 3 | | |
| 438832609 | --- | | | 4 | |
| 438832646 | --- | | | | 5 |

```
#include <arm_itm.h>

......

for (x=0;x<10;x++)
{
  for (y=1;y<5;y++)
  {
    ITM_EVENT8(y,x);
  }
}
```

# Execution time measurement

- Use the functionality of ITM events to measure the time consumption of a piece of code.
- Send two ITM packets before and after the code to be measured and the actual execution time is the interval between them.
- Easy, accurate and no additional equipments are required!

Event Log

| Cycles | Program Counter | ITM1 | ITM2 | ITM3 | ITM4 |
|---|---|---|---|---|---|
| 286540110 | --- | 0x00000055 | | | |
| 373218529 | --- | 0x000000AA | | | |
| 382817996 | --- | 0x00000055 | | | |
| 476367457 | --- | 0x000000AA | | | |
| 485966926 | --- | 0x00000055 | | | |
| 580480993 | --- | 0x000000AA | | | |
| 590080464 | --- | 0x00000055 | | | |

```
#include <arm_itm.h>
……
ITM_EVENT8(1, 0x55);
CodeToBeMeasured( );
ITM_EVENT8(1, 0xAA);
```

- Time consumption of CodeToBeMeasured() on a 100MHz CPU:
  - (476367457 – 382817996) / 100000000 = 0.935 (s)

# Instruction trace

- Collect a sequence of every executed instruction continuously for a selected portion of the program.
- Developers can inspect the program flow up to a specific state and locate the origin of the problem.
- Very useful for locating errors that have irregular symptoms and occur sporadically.
    - Illegal instructions and data aborts
    - Runaway programs
    - Interrupt/exception problems
    - Context switch problems
    - ……
- Also helpful for analyzing dynamic system behaviors
    - Code profiling
    - Code coverage

# Instruction trace techniques

- ETM (Embedded Trace Macrocell)
  - Off-chip trace buffer (in the probe, 2~32 MB)
  - 4~16-bit data bus at CPUCLK or CPUCLK/2
  - High requirement on the board design, e.g. for acceptable signal quality
  - Expensive trace probes required (e.g. JTAGjet-Trace)

- ETB (Embedded Trace Buffer)
  - On-chip dedicated trace buffer (small – a few Kbytes)
  - No extra pins, no requirement for trace probes
  - Cortex-M3, Cortex-M4

- MTB (Micro Trace Buffer)
  - On-chip configurable trace buffer (small – a few Kbytes)
  - No extra pins, no requirement for trace probes
  - Cortex-M0+

# Collect executed instructions

- Go to “ETM Trace” window to check the recorded instructions together with mixed C source code.

*Trace Enable/Disable*

*Trace Packet Number*

*Cycles Counter*

*Instruction Address*

*Source/Disassembly*

ETM Trace

| # | Cycles | Address | Trace | | Exec |
|---|---|---|---|---|---|
| 918 | 48819 | 0x00002754 | MSR | PRIMASK, R0 | Thumb |
| | | | BX | LR | |
| 919 | – | 0x00002758 | BX | LR | Thumb |
| | | | } | | |
| 920 | 48822 | 0x000035d2 | POP | {R0, R4-R7, PC} | Thumb |
| | | | if ((test % 7) == 0) | | |
| 921 | 48825 | 0x00003d4c | LDR.N | R0, ??DataTable5_2 ... | Thumb |
| 922 | – | 0x00003d4e | LDR | R0, [R0] | Thumb |
| 923 | – | 0x00003d50 | MOVS | R1, #7 | Thumb |
| 924 | – | 0x00003d52 | UDIV | R2, R0, R1 | Thumb |
| 925 | | 0x00003d56 | MLS | R0, R1, R2, R0 | Thumb |
| 926 | – | 0x00003d5a | CMP | R0, #0 | Thumb |
| 927 | – | 0x00003d5c | BNE.N | ??App_Task_LED4_1 ... | Thumb |
| | | | printf("hello"); | | |
| 928 | 48828 | 0x00003d76 | LDR.N | R0, ??DataTable5_10... | Thumb |

ETM Trace | ETM Function Trace

# View the trace data at function-level

- Go to “ETM Function Trace” window to view function-level information.
- Useful to find the internal process of complex functions, or the actual calling sequence of interrupt / task switches.

*Trace Enable/Disable*

*Trace Packet Number*

*Cycles Counter*

*Function Address*

*Function Name*

ETM Function Trace

| # | Cycles | Address | Trace | Exec |
|---|---|---|---|---|
| 1318 | 98782 | 0x00003d20 | App_Task_LED4 + 16 | Thumb |
| 1327 | 146788 | 0x0000423c | printf | Thumb |
| 1339 | 146793 | 0x00000410 | _PrintfFull | Thumb |
| 1357 | 146803 | 0x000049c4 | _Prout | Thumb |
| 1362 | 146807 | 0x0000488c | putchar | Thumb |
| 1372 | 146810 | 0x000048b4 | __write | Thumb |
| 1376 | 146813 | 0x000048c4 | __dwrite | Thumb |
| 1382 | 146820 | 0x00004840 | __iar_sh_stdout_swo | Thumb |
| 1407 | 146833 | 0x000048da | __dwrite + 22 | Thumb |
| 1408 | 146835 | 0x000048be | __write + 10 | Thumb |
| 1409 | 146837 | 0x000048a6 | putchar + 26 | Thumb |
| 1413 | 146842 | 0x000049d0 | _Prout + 12 | Thumb |
| 1418 | 146847 | 0x00000432 | _PrintfFull + 34 | Thumb |
| 1430 | 146857 | 0x00004256 | printf + 26 | Thumb |
| 1432 | 146862 | 0x00003d50 | App_Task_LED4 + 64 | Thumb |

ETM Trace | ETM Function Trace

# Using trace start/stop breakpoints

**Right-click in the source or disassembly window:**

- Move to PC
- Run to Cursor
- Code Coverage ▸
- Instruction Profiling ▸
- Toggle Breakpoint (Code)
- Toggle Breakpoint (Log)
- **Toggle Breakpoint (Trace Start)**
- Toggle Breakpoint (Trace Stop)
- Toggle Breakpoint (Trace Filter)
- Enable/Disable Breakpoint
- Edit Breakpoint...
- Set Next Statement

**Edit Breakpoint**

Trace Start

Trigger at: {C:\Test\LCD_Demo\app\main.c}.163.3 [Edit ...]

Access type: Read/Write, Read, Write, Fetch

Size: Auto (1), Manual 1

Trigger range
Requested: 0x0800BC94 - 0x0800BC94
Effective: 0x0800BC94 - 0x0800BC94
Extend to cover requested range

Match data: Enable, Value: 0x00000000, Mask: 0xFFFFFFFF

OK Cancel

**The breakpoint is triggered when the instruction at the specified address is fetched.**

- Go to Source
- **Edit...**
- Delete
- Disable
- Enable All
- Disable All
- Delete All
- New Breakpoint ▸

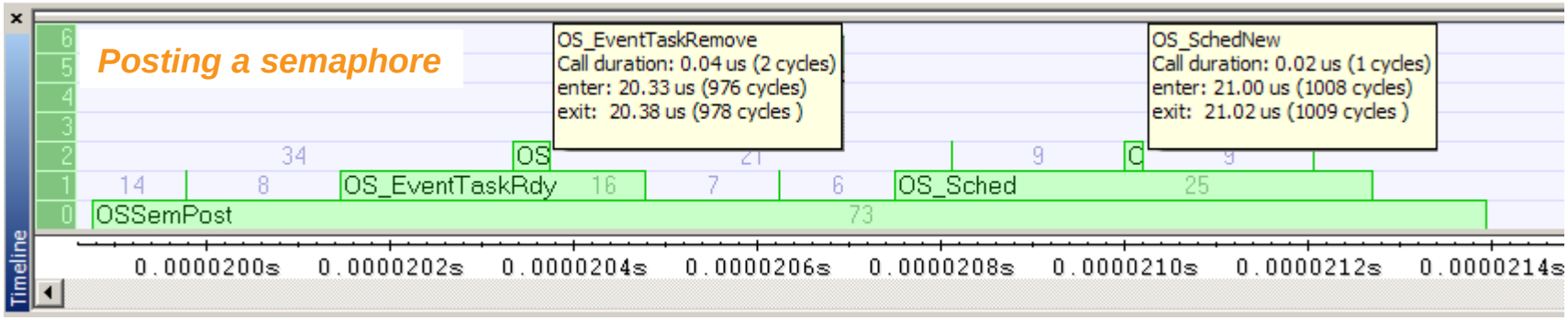
**Check the status and edit the properties of the breakpoint in View → Breakpoints:**

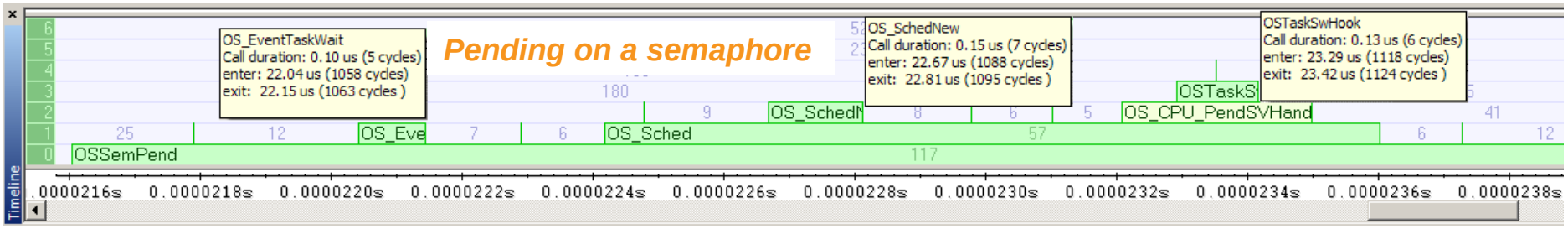
Breakpoints

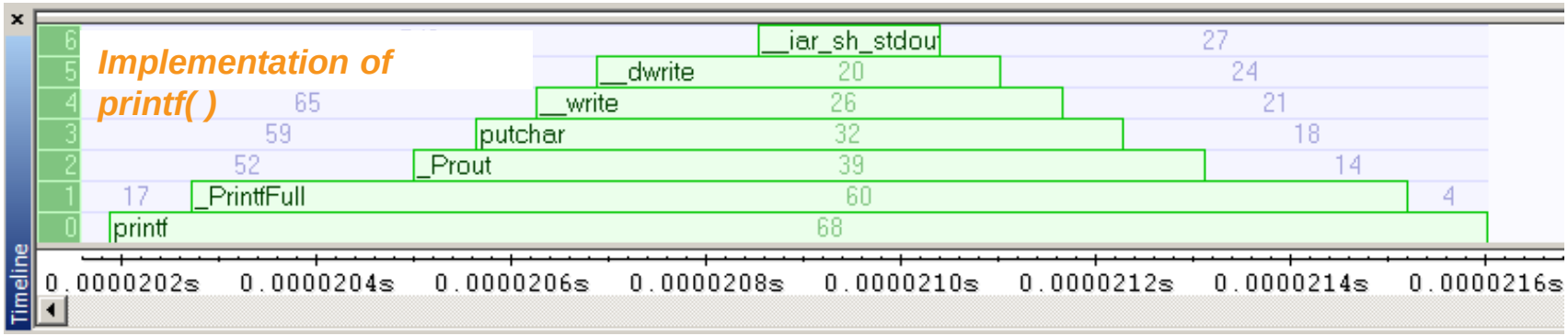
| Breakpoint |
|---|
| Trace Start @ main.c:163.3 [Fetch] [0x0800BC94] |

**location in the source** — **address of the instruction**

# Graphical call stack



# Code coverage

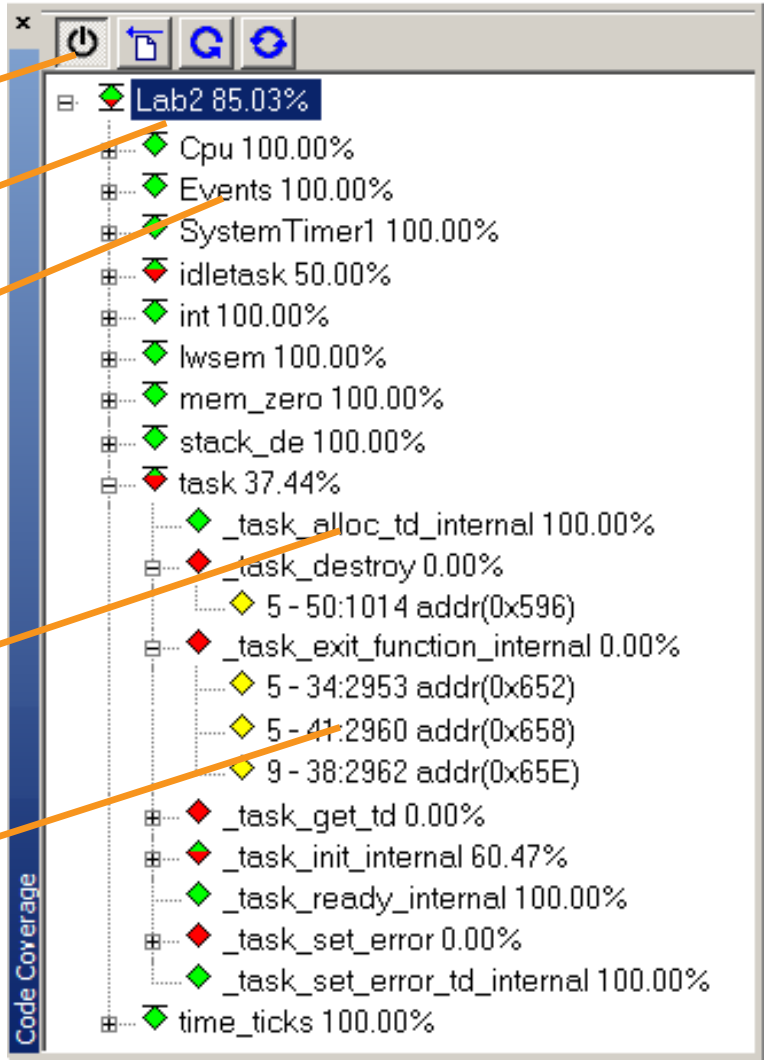
Enable/Disable

Project Name

Module Name

Function Name

Column/Line Number




- *Red*
  *0% of the module or function has been executed.*
- *Green*
  *100% of the module or function has been executed.*
- *Red & Green*
  *Some part of the module or function has been executed.*
- *Yellow*
  *The statement in C/C++ source code that has not been executed.*

# EWARM debugging environment

K60N512 - IAR Embedded Workbench IDE - ARM 7.70.2

File Edit View Project Debug Disassembly I-jet/JTAGjet µC/OS-II Tools Window Help

Workspace: Debug
Files: K60N512 - Debug
APP: app.c, app_cfg.h, app_hooks.c, cpu_cfg.h

app.c os_core.c

```
1775        p_arg = p_arg;
1776        for (;;) {
1777            OS_ENTER_CRITICAL();
1778            OSIdleCtr++;
1779            OS_EXIT_CRITICAL();
1780            OSTaskIdleHook();
1781        }
1782    }
1783    /*$PAGE*/
1784    /*
```

Disassembly

```
OS_CPU_SR  cpu_sr = 0u;
0x2460: 0x2500          MOVS    R5, #0
OS_ENTER_CRITICAL();
0x2462: 0xf000 0xf97b   BL      OS_CPU_
0x2466: 0x0005          MOVS    R5, R0
OSIdleCtr++;
0x2468: 0x48a5          LDR.N   R0, [PC
```

Register: Current CPU Registers

| | | | |
|---|---|---|---|
| R0 | 0x07070707 | R9 | 0x09090909 |
| R1 | 0x20002A48 | R10 | 0x10101010 |
| R2 | 0x02020202 | R11 | 0x11111111 |
| R3 | 0x03030303 | R12 | 0x12121212 |
| R4 | 0x00000000 | APSR | 0x40000000 |
| R5 | 0x00000000 | IPSR | 0x00000000 |
| R6 | 0x06060606 | EPSR | 0x01000000 |
| R7 | 0x07070707 | PC | 0x00002462 |
| R8 | 0x08080808 | SP | 0x20002000 |

Timeline: testPoint1, PendSV, SysTick, INT_UART3_RX_TX, Linear ITrgPwr [mA], ITM1

testPoint1 @ 0x200029F4 + ?
SWO capture clock (48 MHz)

Value is 5 when PC is 0x00003C8A
Access type: Write
Access time: 9s 989518.98 us (479496911 cycles)

Current max: 9 at 3s 589019.15 us (172272919 cycles)
Current min: 0 at 3s 989050.29 us (191474414) cycles)

Function Profiler

| Function | PC Samples | PC Samples (%) | Address |
|---|---|---|---|
| OS_TaskIdle | 80243 | 49.74 | 0x245c–0x247d |
| OSTaskIdleHook | 38694 | 23.98 | 0x2852–0x2859 |
| OS_CPU_SR_Save | 20498 | 12.71 | 0x275c–0x2763 |
| App_TaskIdleHook | 12584 | 7.80 | 0x34a8–0x34a9 |
| OS_CPU_SR_Restore | 7651 | 4.74 | 0x2764–0x2769 |
| OSTimeTick | 493 | 0.31 | 0x1eea–0x1fdf |
| OSIntExit | 427 | 0.26 | 0x1d2c–0x1dbf |
| OSTimeTickHook | 373 | 0.23 | 0x2908–0x292b |
| OSTaskStkChk | 154 | 0.10 | 0x32fe–0x338f |
| OS_TaskStatStkChk | 51 | 0.03 | 0x251a–0x255d |
| OSSemPend | 34 | 0.02 | 0x3506–0x35e7 |
| OS_EventTaskRdy | 17 | 0.01 | 0x1fe2–0x209b |
| OS_EventTaskWait | 17 | 0.01 | 0x209c–0x2129 |
| OS_SchedNew | 14 | 0.01 | 0x243c–0x245b |

Memory

```
00000000  90 2e 00 20 31 4a 00 00 db 11 00 00 dd 11 00 00   ... 1J..........
00000010  b3 36 00 00 df 11 00 00 e1 11 00 00 e3 11 00 00   .6..............
00000020  e3 11 00 00 e3 11 00 00 e3 11 00 00 e3 11 00 00   ................
00000030  e3 11 00 00 e3 11 00 00 9f 27 00 00 2d 29 00 00   .........'..-)..
00000040  bd 36 00 00 c7 36 00 00 d1 36 00 00 db 36 00 00   .6...6...6...6..
00000050  e5 36 00 00 ef 36 00 00 f9 36 00 00 03 37 00 00   .6...6...6...7..
00000060  0d 37 00 00 17 37 00 00 21 37 00 00 2b 37 00 00   .7...7..!7..+7..
```

Task List

| Name | Ref | Prio | State |
|---|---|---|---|
| LED Orange Task | 4 | 3 | Sem |
| LED Yellow Task | 5 | 4 | Sem |
| LED Green Task | 6 | 5 | Sem |
| LED Blue Task | 7 | 6 | Dly |
| uC/OS-II Tmr | 2 | 61 | Sem |
| uC/OS-II Stat | 1 | 62 | Dly |

Status: Running µC/OS-II RTOS V292.07

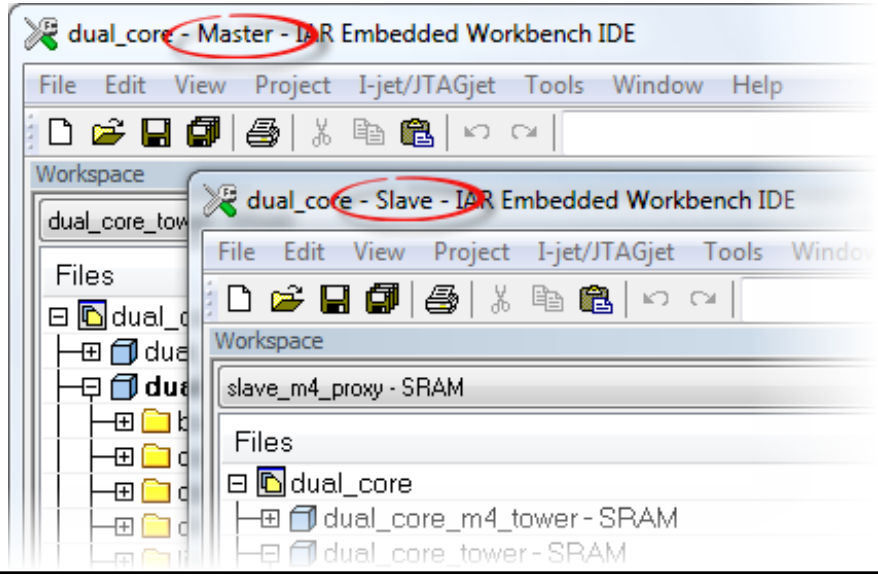
Statistics: Ready
CPU Usage: 1%
Tasks: 8
Idle Counter: 110174
Context Switches: 1504

Nesting
Interrupt: 0
Multitask Lock: 0
Step Mode: Disabled

Ready | Ln 1783, Col 10 | System

# Multi-core debugging

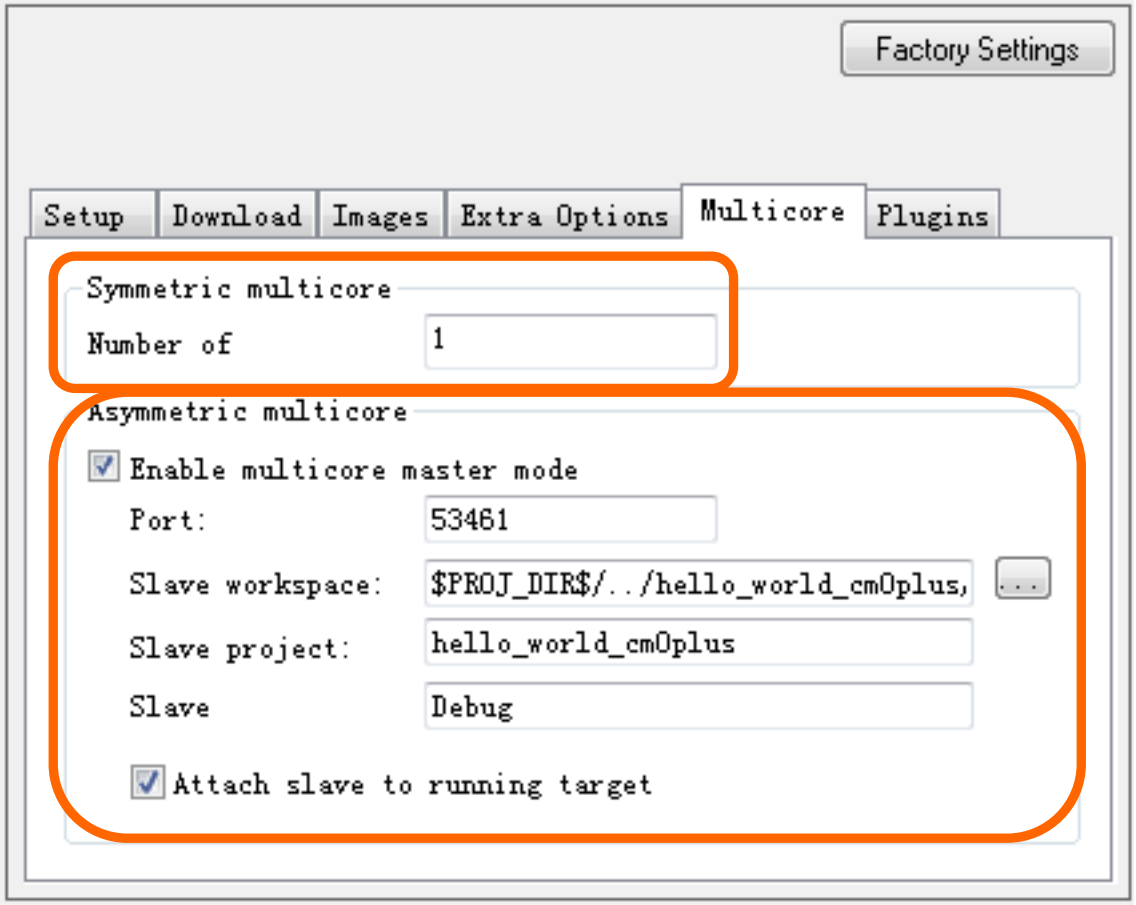
- Symmetric multicore debugging (SMP)
  - Debugging two or more identical cores;
  - One single debug probe;
  - One single instance of IAR EWARM IDE.
- Asymmetric multicore debugging (AMP)
  - Debugging two different cores;
  - One single debug probe;
  - Two cooperating instances of IAR EWARM IDE.



# Multi-core debugging: Configuration

**SMP:**
**Number of cores**

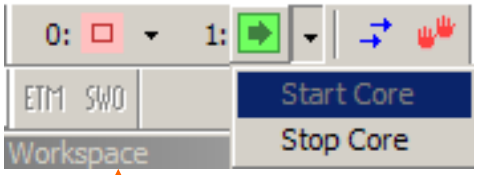
**AMP:**
**Set the direction to the slave project in the master project.**



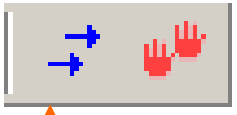
# Multi-core debugging: LPC54114

**Execution State :**

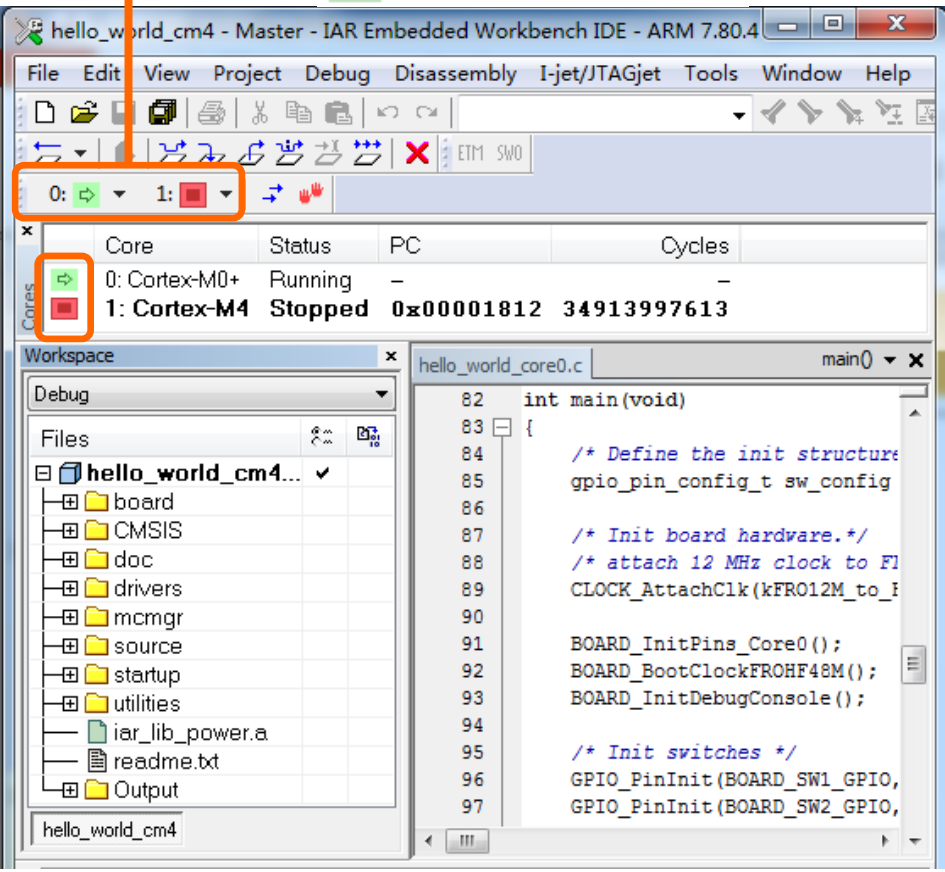
- in focus, not executing
- not in focus, not executing
- in focus, executing
- not in focus, executing
- in focus, in sleep mode
- not in focus, in sleep mode

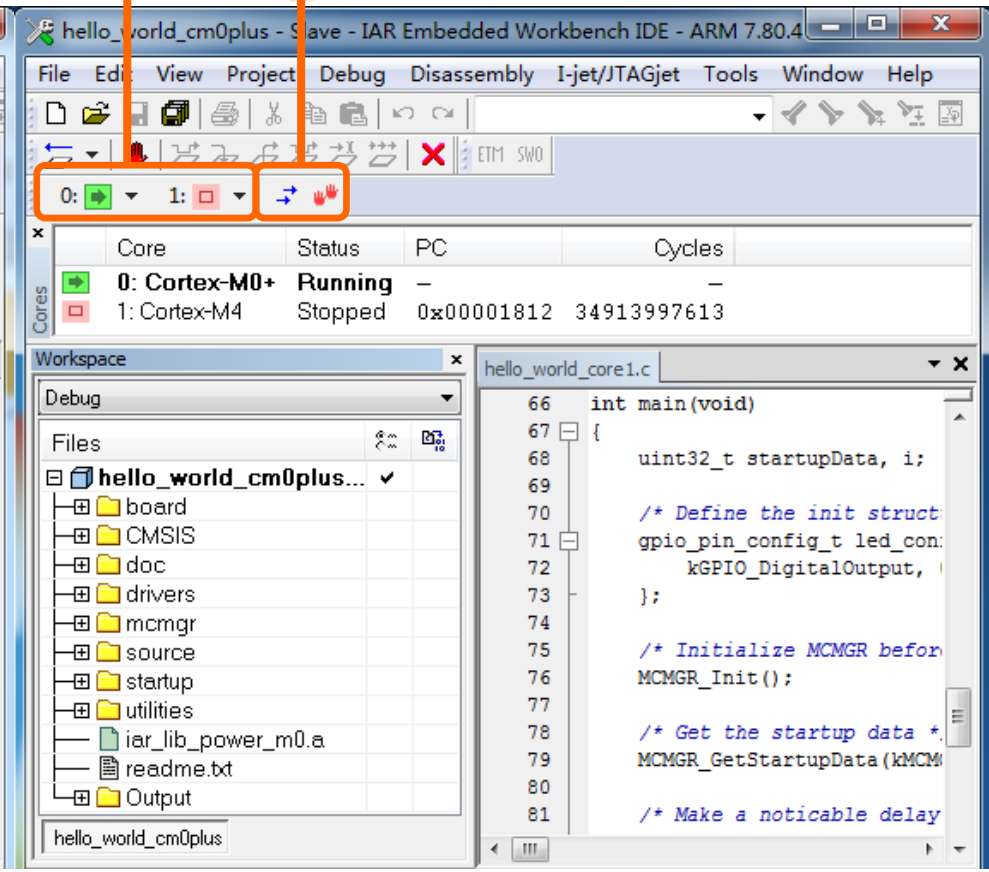



**Execute command for each core**



**Start all cores**
**Stop all cores**



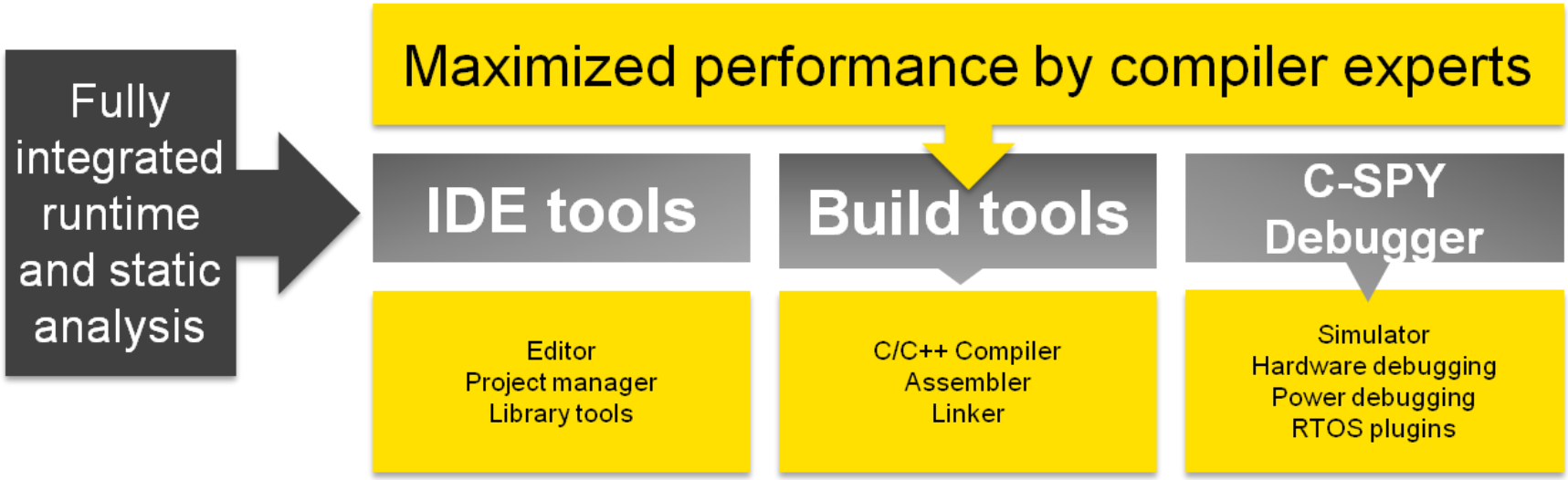



# Keeping Safe at C:
# Static & Runtime Code Analysis

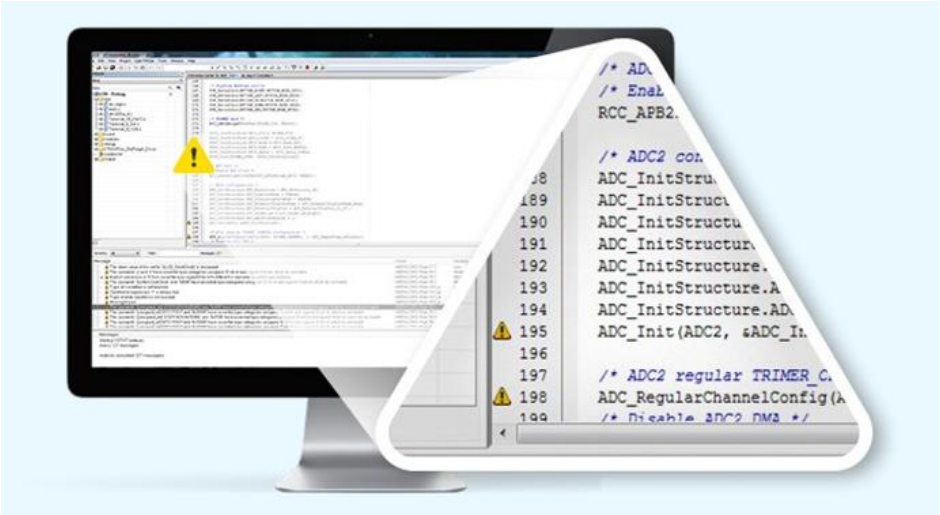
IAR SYSTEMS

# Integrated code analysis add-on tools

- Static code analysis: C-STAT
  - Analyze the C/C++ source code without executing the program.
  - Fully integrated in IAR Embedded Workbench for ARM.
- Runtime code analysis: C-RUN
  - Find C programming errors at runtime.
  - Fully integrated in IAR Embedded Workbench for ARM.



# C-STAT and C-RUN



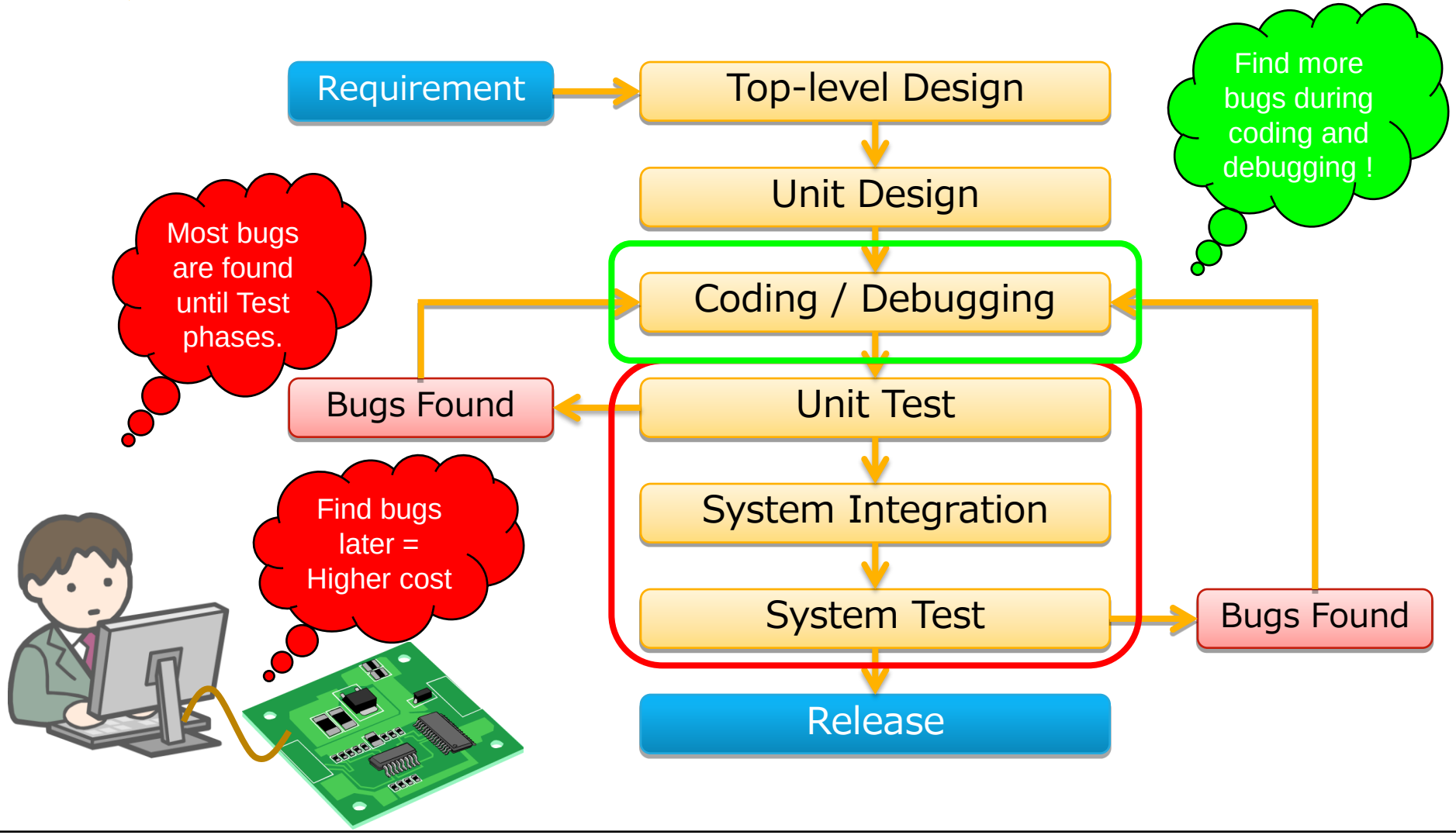

## C-STAT Static analysis

C-STAT performs advanced analysis of your C/C++ code and finds potential issues. It helps you improve your code quality as well as prove alignment with standards such as MISRA C:2012.

## C-RUN Runtime analysis

C-RUN helps you find errors at an early stage. It is completely integrated with IAR Embedded Workbench for ARM, and provides detailed runtime error information.

# Code analysis: Find bugs earlier



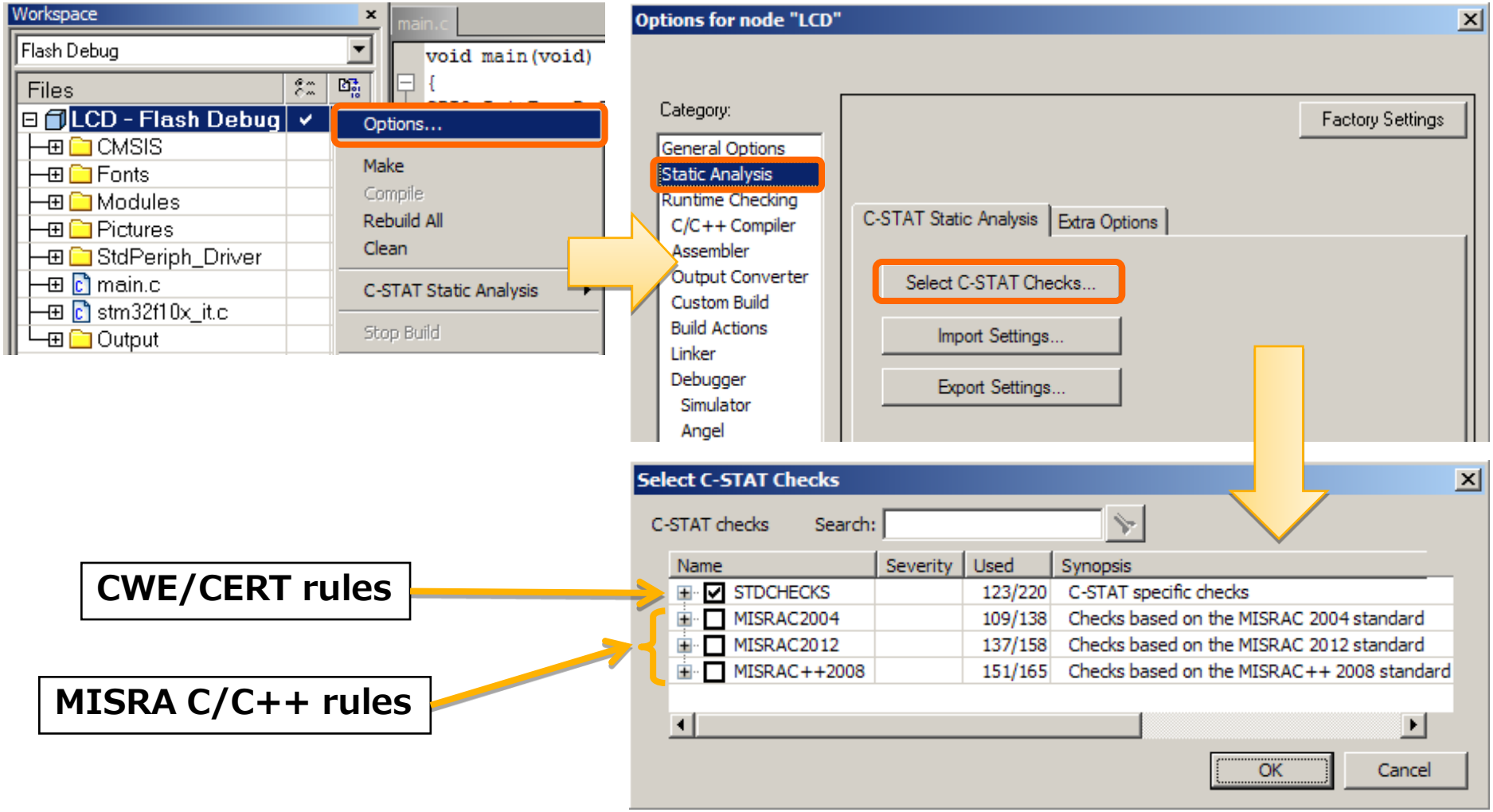
# C-STAT: What does it check

- Common Weakness Enumeration
- cwe.mitre.org
- An unified and measurable set of software weaknesses.
- Enumerate design and architecture weaknesses, as well as low-level coding errors.

- Computer Emergency Response Team
- www.cert.org
- C/C++ secure coding standards, identifying insecure constructs which could expose a weakness or vulnerability in the software.
- Guidelines to avoid implementation, coding as well as low-level design errors.

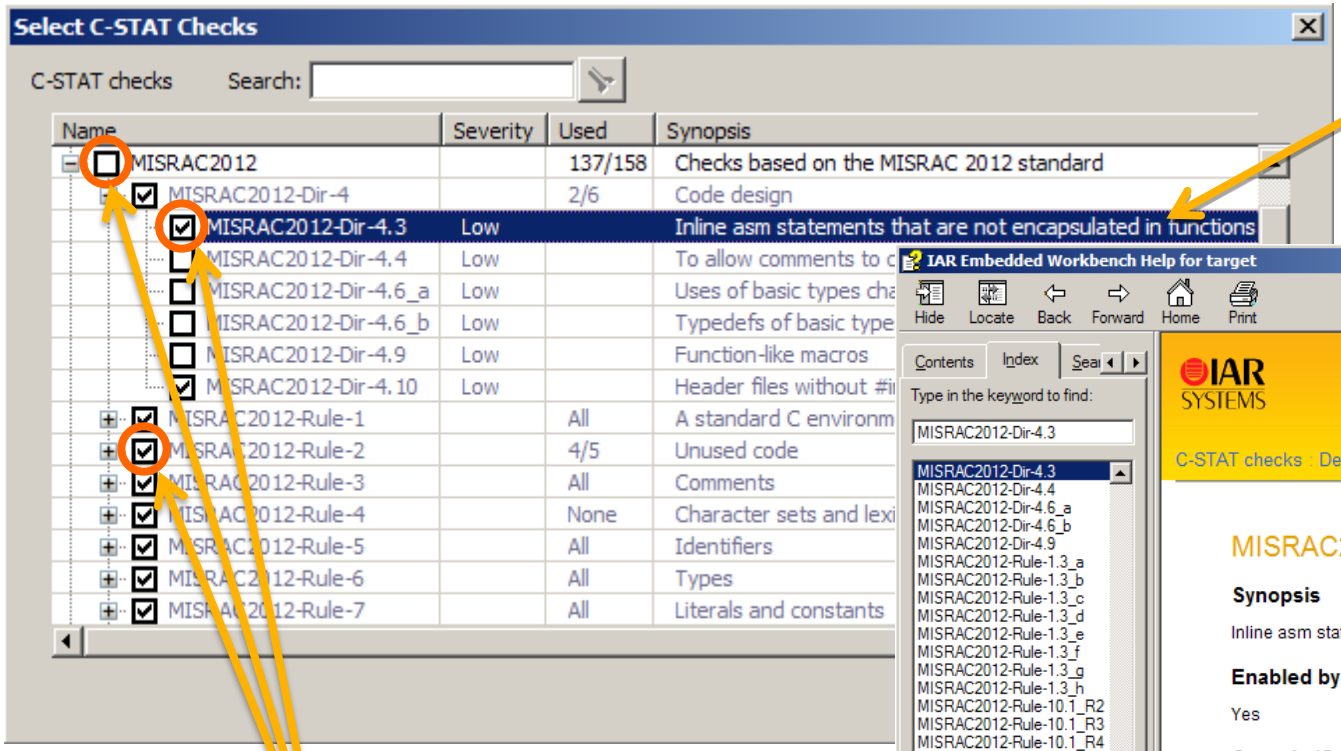
- Motor Industry Software Reliability Association
- www.misra.org.uk
- MISRA C:2004 (MISRA C2): Identify unsafe code constructs in the C89 standard.
- MISRA C:2012 (MISRA C3): Extend the support to C99 version of the programming language whilst maintaining the guidelines for C89 standard.
- MISRA C++:2008: Identify unsafe code constructs in the 1998 C++ standard.

# C-STAT: Options in IAR EWARM

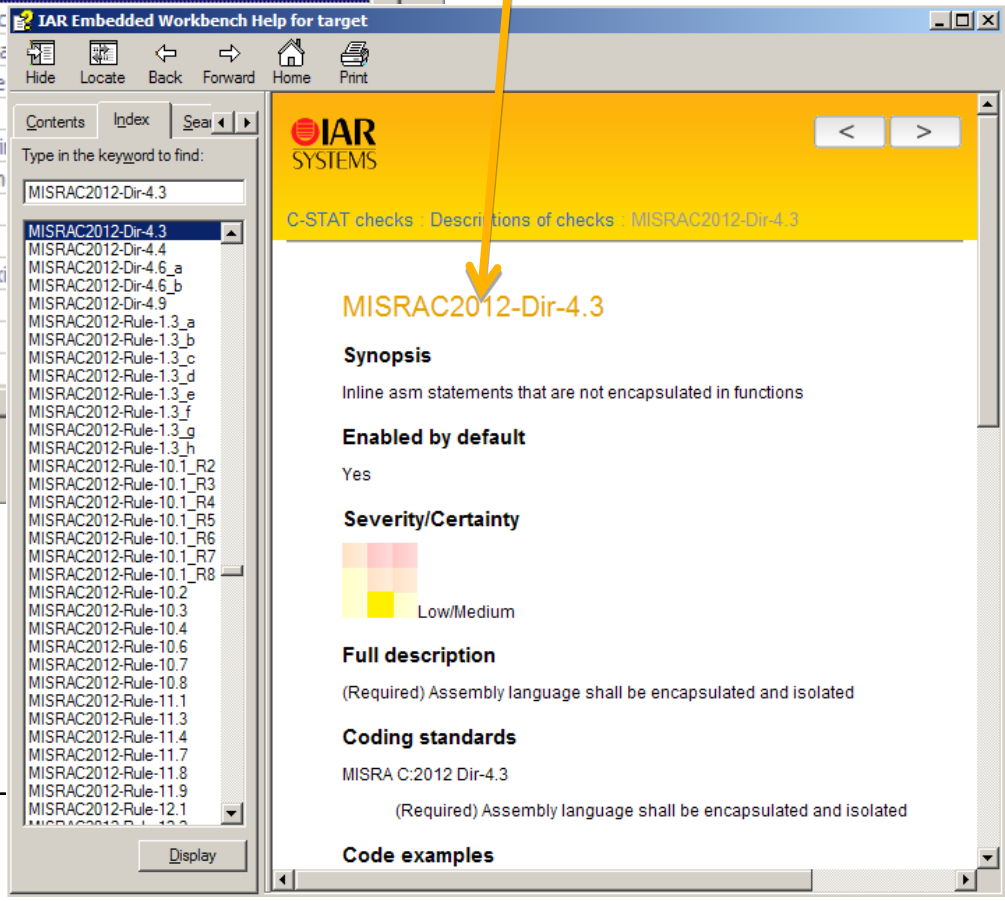


# C-STAT: Rules configuration

IAR SYSTEMS




*Highlight a rule and press F1 to show the detailed description.*




*Enable or disable a set of rules or any individual rule.*

# C-STAT: Result of analysis

IAR SYSTEMS

*Filter the C-STAT messages by selecting a level of severity: All, Low, Medium or High.*

Workspace

Flash Debug

Files

- LCD - Flash Debug
  - CMSIS
  - Fonts
  - Modules
  - Pictures
  - StdPeriph_Driver
  - main.c
  - stm32f10x_it.c
  - Output

LCD

stm32f10x_adc.c

```
if (Rank < 7)
{
  /* Get the old register value */
  tmpreg1 = ADCx->SQR3;
  /* Calculate the mask to clear */
  tmpreg2 = SQR3_SQ_Set << (5 * (Rank - 1));
  /* Clear the old SQx bits for the selected rank */
  tmpreg1 &= ~tmpreg2;
  /* Calculate the mask to set */
  tmpreg2 = (uint32_t)ADC_Channel << (5 * (Rank - 1));
  /* Set the SQx bits for the selected rank */
  tmpreg1 |= tmpreg2;
  /* Store the new register value */
```

Severity: All  Filter:  Messages: 31

| Message | Check | Severity | File | |
|---|---|---|---|---|
| drv_glcd.c (19 messages) | | | drv_glcd. | |
| glcd_ll.c (2 messages) | | | glcd_ll.c | |
| iar_logo.c (1 message) | | | iar_logo.c | |
| main.c (1 message) | | | main.c | |
| misc.c (1 message) | | | misc.c | |
| stm32f10x_adc.c (2 messages) | | | stm32f10x_adc.c | |
| RHS argument is in interval [-5,25] which is out of range of the shift operator | ATH-shift-bounds | Medium | stm32f10x_adc.c | 635 |
| RHS argument is in interval [-5,25] which is out of range of the shift operator | ATH-shift-bounds | Medium | stm32f10x_adc.c | 639 |
| stm32f10x_gpio.c (1 message) | | | stm32f10x_gpio.c | |
| stm32f10x_it.c (1 message) | | | stm32f10x_it.c | |
| Terminal_18_24x12.c (1 message) | | | Terminal_18_24x12.c | |
| Terminal_6_8x6.c (1 message) | | | Terminal_6_8x6.c | |
| Terminal_9_12x6.c (1 message) | | | Terminal_9_12x6.c | |

C-STAT Messages

*Double click the C-STAT message to direct to the line of source code.*

IAR Embedded Workbench Help for target

Hide Locate Back Forward Home Print

Contents Index Search

Type in the keyword to find: ATH-shift-bounds

ATH-shift-bounds
ATH-shift-neg
ATH-sizeof-by-sizeof
atof
atoi
atol
atoll
atomic operation
Attach to program (debugger option)
attributes
object
type
attributes on sections, definition of
Auto (Default breakpoint type setting)
Auto (JTAG/SWD speed setting)
Auto (Language setting)
Auto (Probe config setting)
Auto (SWO protocol setting)
Auto (SWO setting)
Auto code completion and parameter h
Auto indent (editor option)
Auto Scroll (Timeline window context m
auto variables
at function entrance
programming hints for efficient code
using in inline assembler statements
Auto window
auto, packing algorithm for initializers
auto_ptr
~auto_ptr
auto_ptr::~auto_ptr
auto_ptr::auto_ptr
auto_ptr::element_type
auto_ptr::get
auto_ptr::operator auto_ptr_ref<Other>
auto_ptr::operator auto_ptr<Other>
auto_ptr::operator*
auto_ptr::operator=
auto_ptr::operator->
auto_ptr::release

Display

IAR SYSTEMS

C-STAT checks : Descriptions of checks : ATH-shift-bounds

## ATH-shift-bounds

**Synopsis**

Out of range shifts

**Enabled by default**

Yes

**Severity/Certainty**

Medium/Medium

**Full description**

A shift operator on an n-bit argument may only shift between 0 and n−1 bits. In this case, the right-hand operand may be negative, or too large. This check is for all platforms. The behavior in this situation is undefined; the code may work as intended, or data could become erroneous.

**Coding standards**

CERT INT34-C

Do not shift a negative number of bits or more bits than exist in the operand

*Highlight the C-STAT message and press F1 to show the related rules information.*

# C-RUN: What does it check

Heap checking

Bounds checking

Arithmetic checking

Category:

- General Options
- Runtime Checking
  - C/C++ Compiler
  - Assembler
  - Output Converter
  - Custom Build
  - Build Actions
  - Linker
  - Debugger
    - Simulator
    - Angel
    - CMSIS DAP
    - GDB Server
    - IAR ROM-monitor
    - I-jet/JTAGjet
    - J-Link/J-Trace
    - TI Stellaris
    - Macraigor
    - PE micro
    - RDI
    - ST-LINK

Factory Settings

C-RUN Runtime Checking

- [x] Enable
  - [x] Use checked heap
  - [x] Enable bounds checking
    - Instrumentation
      - [x] Track pointer bounds
        - [x] Check accesses
        - [ ] Generate functions callable from non-instrumented code
      - [ ] Check pointers from non-instrumented functions
    - Global bounds table
      - [ ] Check pointers from non-instrumented memory
      - Number of entries: 1000
  - Insert checks for
    - [x] Integer overflow
      - [ ] Including unsigned
    - [x] Integer conversion
      - [ ] Including explicit casts
    - [x] Integer shift overflow
      - [ ] Including unsigned shifts
    - [x] Division by zero
    - [x] Unhandled switch case

# C-RUN: How does it work

- Traditional runtime analysis tools:
  - Independent with compiler and debugger;
  - Different applications and license models;
  - Less knowledge about the target and optimization;
  - Insert test code at the source code level;
  - Large overhead in memory size and execution speed.
- C-RUN:
  - Created by compiler and debugger experts;
  - Fully integrated within IAR Embedded Workbench;
  - Insert target optimized test code directly during compilation;
  - Replace the C/C++ standard library with a dedicated library which contains special functionality for runtime error checking;
  - Result in minimized ROM/RAM overhead and speed penalty.

# Detecting integer overflow

```
void main (void)
{
  int v1 = 0x7fffffff;
  unsigned int v2 = 0xffffffff;

  v1++; /* signed integer overflow */
  v2++; /* unsigned integer overflow */
}
```

Insert checks for

- [x] Integer overflow
  - [x] Including unsigned
- [ ] Integer conversion
  - [ ] Including explicit casts
- [ ] Integer shift overflow
  - [ ] Including unsigned shifts
- [ ] Division by zero
- [ ] Unhandled switch case

C-RUN Messages

Default action: Stop  Filter:  Messages: 2

| Messages | Source File | PC |
|---|---|---|
| Signed integer overflow | main.c 6:3-6 | 0x000000E8 |
| Unsigned integer overflow | main.c 7:3-6 | 0x00000114 |
| Result is greater than the largest representable number: | | |
| 4294967295 (0xffffffff) + 1 (0x1). | | |
| Call Stack | | |
| main | main.c 7:3-7 | |
| [_call_main + 0x9] | | |

# Detecting heap errors

```c
#include <stdlib.h>
#include <iar_dlmalloc.h>

void main (void)
{
  char *c = malloc(10);
  c = malloc(20);              /* memory leak */

  free(c);

  /* check for memory leaks, manually called */
  __iar_check_leaks();
}
```

- [x] Use checked heap
- [ ] Enable bounds checking

Instrumentation

- [ ] Track pointer bounds
  - [ ] Check accesses
  - [ ] Generate functions callable from non-instrumented code
  - [ ] Check pointers from non-instrumented functions

C-RUN Messages

Default action: Stop  Filter:  Messages: 1

| Messages | Source File | PC |
|---|---|---|
| Memory leak | main.c 12:3-21 | 0x00003B16 |
| There were a total of 1 heap blocks with no references. | | |
| Heap block 0 at 0x00102450 has no references. | | |
| The block was allocated at line 6 of main.c. | main.c 6:13-22 | |
| Call Stack | | |
| main | main.c 13:1-1 | |
| [_main + 0x4] | | |

# Detecting out-of-bounds

```
int main (void)
{
  int i, j;
  int a[3] = {1, 2, 3};

  for (i=0; a[i]!=0; i++)  /* out of bounds */
  {                        /* when i==3     */
    j = a[i];
  }

  return j;
}
```

- [ ] Use checked heap
- [x] Enable bounds checking

Instrumentation

- [x] Track pointer bounds
  - [x] Check accesses
  - [ ] Generate functions callable from non-instrumented code
  - [ ] Check pointers from non-instrumented functions

C-RUN Messages

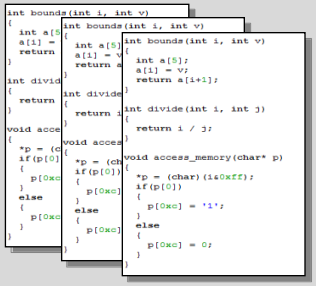
Default action: Stop  Filter:  Messages: 1

| Messages | Source File | PC |
|---|---|---|
| Access out of bounds | main.c 6:13-16 | 0x000001E8 |
| Access outside pointer bounds: | | |
| Access 0x00101ff0 - 0x00101ff4 | | |
| Bounds 0x00101fe4 - 0x00101ff0, int a[3]; | main.c 4:7-7 | |
| Call Stack | | |

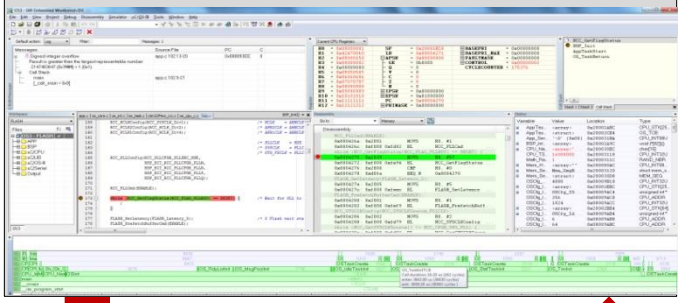
# Take full control of your development

IAR SYSTEMS

## Implement your design in code

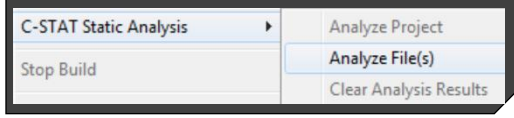


## Build and debug the application

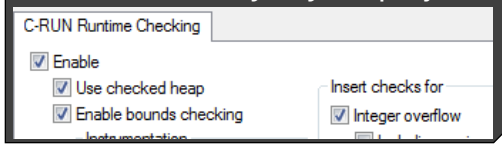


## Release the application

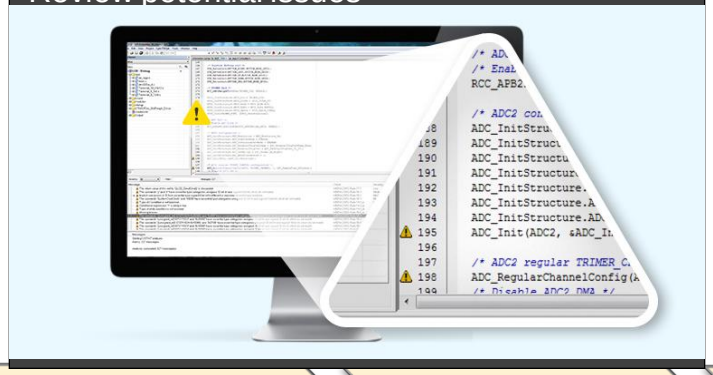
## Let C-STAT analyze your code



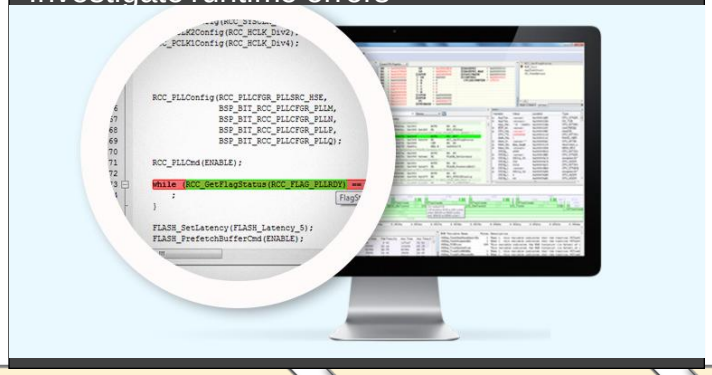

## Let C-RUN analyze your project




## Review potential issues



## Investigate runtime errors



Requirements → Design → Implementation → Verification → Maintenance