

面向功能安全的嵌入式开发工具

IAR Systems, Shanghai
Cynthia Hu



- IAR Systems公司介绍
- IoT嵌入式软件设计
- 功能安全认证
- C语言的安全性讨论
- C-STAT: 静态代码分析工具
- C-RUN: 动态代码分析工具

IAR Systems公司介绍





- IAR Systems公司成立于1983年
- 总部位于瑞典乌普萨拉
- 为全球30多个国家地区提供本地服务
- 支持多达10,000款微控制器
 - 支持3000多款 ARM芯片
- 全球领先的嵌入式开发工具供应商
- 主要产品
 - IAR Embedded Workbench: C/C++ 编译调试工具
 - IAR visualSTATE: 状态机建模和软件设计工具
 - IAR I-jet / I-scope / JTAGjet Trace: 跟踪调试器
- IAR Systems中国办事处:
 - 上海, 021-63758658

IoT嵌入式软件设计特点

IoT嵌入式软件设计特点

IoT嵌入式软件设计特点

RTOS & Middleware

软件设计复杂度较高

安全性需求（开放架构）

可靠性要求高

调试难度大

低功耗要求

实时响应速度

代码尺寸

IAR工具提供的帮助

多种成熟RTOS partner方案和
Debug Awareness技术

建模工具

Functional Safety功能安全认证

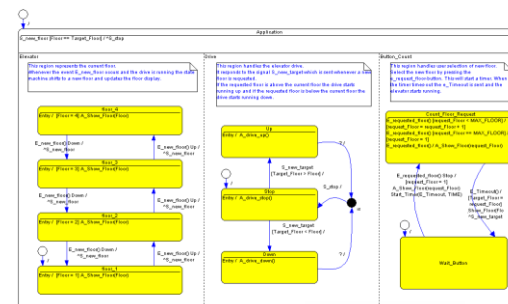
静态分析和运行时分析工具

高级调试功能和Trace功能

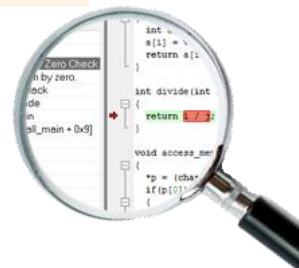
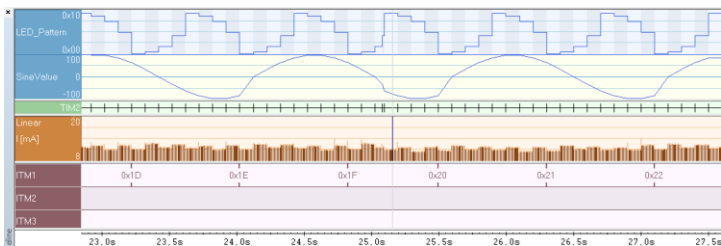
Power debugging技术

速度优化

代码尺寸优化



Address	Type	Value	Comment
0x00000000	CODE	0x00000000	0x00000000
0x00000001	CODE	0x00000001	0x00000001
0x00000002	CODE	0x00000002	0x00000002
0x00000003	CODE	0x00000003	0x00000003
0x00000004	CODE	0x00000004	0x00000004
0x00000005	CODE	0x00000005	0x00000005
0x00000006	CODE	0x00000006	0x00000006
0x00000007	CODE	0x00000007	0x00000007
0x00000008	CODE	0x00000008	0x00000008
0x00000009	CODE	0x00000009	0x00000009
0x0000000A	CODE	0x0000000A	0x0000000A
0x0000000B	CODE	0x0000000B	0x0000000B
0x0000000C	CODE	0x0000000C	0x0000000C
0x0000000D	CODE	0x0000000D	0x0000000D
0x0000000E	CODE	0x0000000E	0x0000000E
0x0000000F	CODE	0x0000000F	0x0000000F
0x00000010	CODE	0x00000010	0x00000010
0x00000011	CODE	0x00000011	0x00000011
0x00000012	CODE	0x00000012	0x00000012
0x00000013	CODE	0x00000013	0x00000013
0x00000014	CODE	0x00000014	0x00000014
0x00000015	CODE	0x00000015	0x00000015
0x00000016	CODE	0x00000016	0x00000016
0x00000017	CODE	0x00000017	0x00000017
0x00000018	CODE	0x00000018	0x00000018
0x00000019	CODE	0x00000019	0x00000019
0x0000001A	CODE	0x0000001A	0x0000001A
0x0000001B	CODE	0x0000001B	0x0000001B
0x0000001C	CODE	0x0000001C	0x0000001C
0x0000001D	CODE	0x0000001D	0x0000001D
0x0000001E	CODE	0x0000001E	0x0000001E
0x0000001F	CODE	0x0000001F	0x0000001F
0x00000020	CODE	0x00000020	0x00000020
0x00000021	CODE	0x00000021	0x00000021
0x00000022	CODE	0x00000022	0x00000022
0x00000023	CODE	0x00000023	0x00000023
0x00000024	CODE	0x00000024	0x00000024
0x00000025	CODE	0x00000025	0x00000025
0x00000026	CODE	0x00000026	0x00000026
0x00000027	CODE	0x00000027	0x00000027
0x00000028	CODE	0x00000028	0x00000028
0x00000029	CODE	0x00000029	0x00000029
0x0000002A	CODE	0x0000002A	0x0000002A
0x0000002B	CODE	0x0000002B	0x0000002B
0x0000002C	CODE	0x0000002C	0x0000002C
0x0000002D	CODE	0x0000002D	0x0000002D
0x0000002E	CODE	0x0000002E	0x0000002E
0x0000002F	CODE	0x0000002F	0x0000002F
0x00000030	CODE	0x00000030	0x00000030
0x00000031	CODE	0x00000031	0x00000031
0x00000032	CODE	0x00000032	0x00000032
0x00000033	CODE	0x00000033	0x00000033
0x00000034	CODE	0x00000034	0x00000034
0x00000035	CODE	0x00000035	0x00000035
0x00000036	CODE	0x00000036	0x00000036
0x00000037	CODE	0x00000037	0x00000037
0x00000038	CODE	0x00000038	0x00000038
0x00000039	CODE	0x00000039	0x00000039
0x0000003A	CODE	0x0000003A	0x0000003A
0x0000003B	CODE	0x0000003B	0x0000003B
0x0000003C	CODE	0x0000003C	0x0000003C
0x0000003D	CODE	0x0000003D	0x0000003D
0x0000003E	CODE	0x0000003E	0x0000003E
0x0000003F	CODE	0x0000003F	0x0000003F





Alliances in a smarter world

All relations are important to us; either they are with customers, partners or investors. With strategic alliances in the connected world, we are showing that we want do more as well as invest more in technology and offer more to our customers.

ALLIANCES

IAR Embedded Workbench for ARM



IDE

Editors

```

void t1()
{
    /* Increment the thread counter. */
    thread_0_counter++;
    if (!!(thread_0_counter%3)) {
        SPI2_SDR0(LDR);
        dataLog=1;
    } else {
        SPI2_SDR0(LDR);
        dataLog=0;
    }
}

/* Sleep for 30 ticks. */
tx_thread_sleep(30);

/* Get event flag 0 to makeup thread 5. */
status = tx_event_flags_get(Event_Flag_0_0x2, TX_00);

/* Check status. */
if (status != TX_SUCCESS)
    break;
}
    
```

Source code control systems

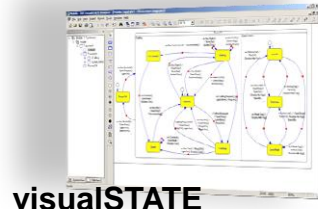
P L U G I N S

IAR Embedded Workbench IDE

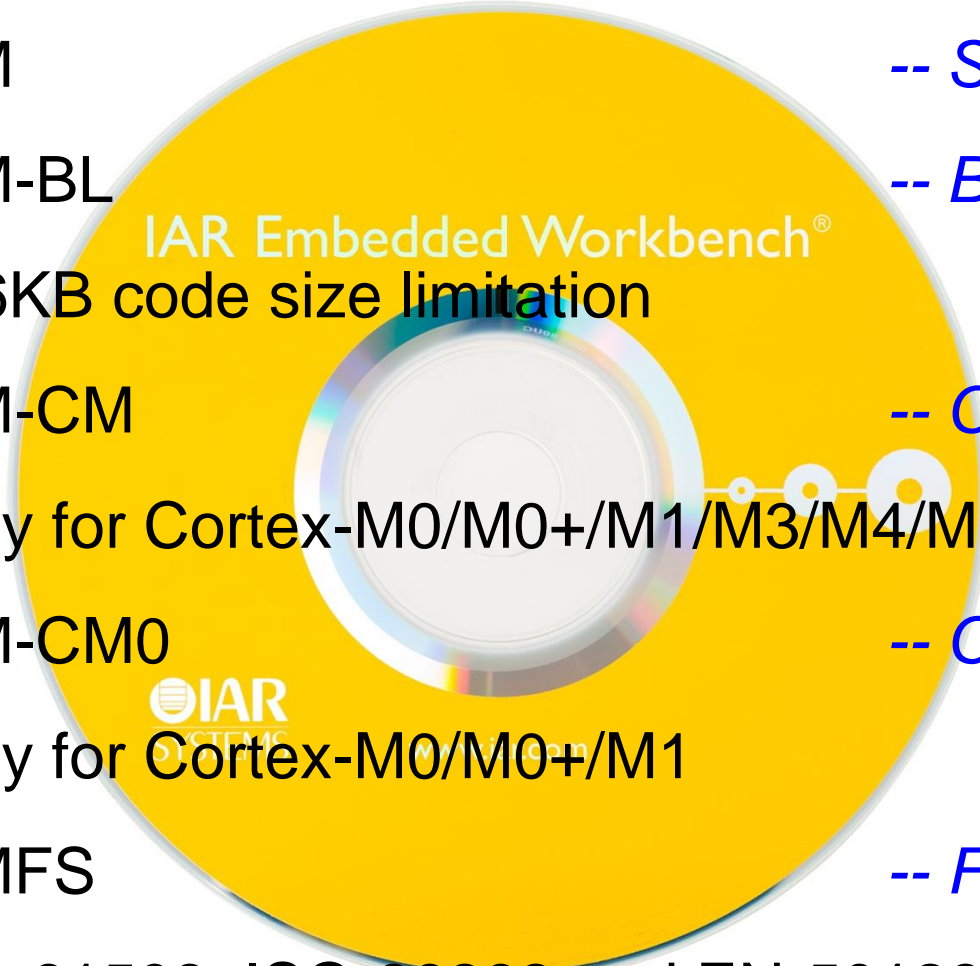
IDE tools	Build tools	IAR C-SPY Debugger
Editor	IAR C/C++ Compiler	Simulator driver
Project manager	Assembler	Hardware system drivers
Library builder	Linker	Power debugging
Librarian		RTOS plug-ins



Configuration tools



EWARM版本分类

- 
- EWARM -- *Standard edition*
 - EWARM-BL -- *Baseline edition*
 - 256KB code size limitation
 - EWARM-CM -- *Cortex-M edition*
 - Only for Cortex-M0/M0+/M1/M3/M4/M7
 - EWARM-CM0 -- *CM0/CM1 only*
 - Only for Cortex-M0/M0+/M1
 - EWARMFS -- *Functional Safety*
 - IEC-61508, ISO-26262 and EN-50128

功能安全认证

什么是功能安全？

- 所谓功能安全，简要说来，就是嵌入式系统在面临以下情况时，是否能够执行正确的操作并准确响应：
 - 错误的输入
 - 硬件故障
- 一般来说，以下这些行业对功能安全有高度需求：
 - 汽车行业
 - 航天电子
 - 医疗电子
- 当然，其他行业的应用方案也受益于功能安全设计

功能安全标准



**Table A.3 – Software design and development –
support tools and programming language**

(See 7.4.4)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Suitable programming language	C.4.5	HR	HR	HR	HR
2	Strongly typed programming language	C.4.1	HR	HR	HR	HR
3	Language subset	C.4.2	---	---	HR	HR
4a	Certified tools and certified translators	C.4.3	R	HR	HR	HR
4b	Tools and translators: increased confidence from use	C.4.4	HR	HR	HR	HR
NOTE 1 See Table C.3.						
NOTE 2 The references (which are informative, not normative) "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.						
* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.						

功能安全标准

- IEC 61508
- ISO 26262
- EN 50128

功能安全认证版本提供

- 功能安全证书
- 认证报告
- 安全手册

可靠的支持和升级服务

- 涵盖整个产品生命周期
- 高优先级的技术支持服务
- 经过认证的服务升级包
- 已知问题报告

The screenshot displays the IAR Embedded Workbench IDE interface. Key components include:

- CSTACK:** A table showing memory stack locations, data, variables, and values.
- Current CPU Registers:** A table listing registers R0 through R9, LPSR, and EPSR with their current values.
- Expression:** A table showing expressions, values, locations, and types for variables like thread_0_counter through thread_7_counter.
- Disassembly:** A window showing assembly code with instructions like LDR, STR, and MOV, along with their addresses and operands.
- Debug:** A window showing the source code with line numbers and comments.
- Execution Profile:** A table showing execution metrics for different threads, including total time, execution percentage, and individual thread times.

Overlaid on the screenshot is the TÜV SÜD Functional Safety logo, which is an octagonal badge with the text "Safety tested", "Production monitored", "TÜV SÜD", and "Functional Safety".

- EWARMFS

- IAR Embedded Workbench for ARM 功能安全认证版本

- 通过TÜV SÜD认证

- 功能安全认证标准

- IEC 61508-3:2010 (SIL 3)

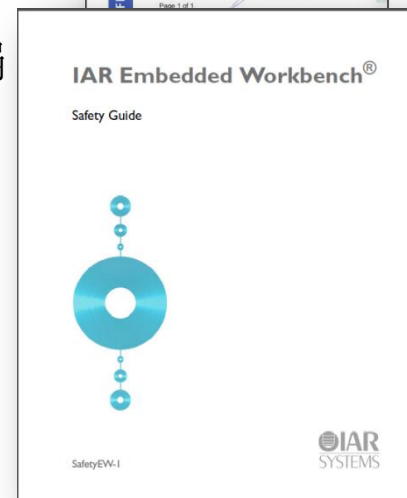
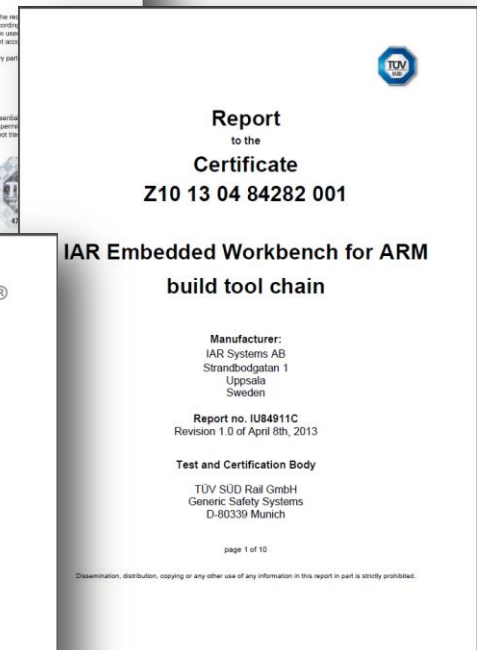
- 面向所有行业的电子，电气以及可编程系统

- ISO 26262-8:2011 (ASIL D)

- 由IEC 61508标准衍生出来的面向道路车辆的安全标准

- EN 50128

- 铁路控制和保护系统安全标准



C语言安全性的讨论

**Table A.3 – Software design and development –
support tools and programming language**

(See 7.4.4)

Technique/Measure *		Ref.	SIL 1	SIL 2	SIL 3	SIL 4
1	Suitable programming language	C.4.5	HR	HR	HR	HR
2	Strongly typed programming language	C.4.1	HR	HR	HR	HR
3	Language subset	C.4.2	---	---	HR	HR
4a	Certified tools and certified translators	C.4.3	R	HR	HR	HR
4b	Tools and translators: increased confidence from use	C.4.4	HR	HR	HR	HR
NOTE 1 See Table C.3.						
NOTE 2 The references (which are informative, not normative) "B.x.x.x", "C.x.x.x" in column 3 (Ref.) indicate detailed descriptions of techniques/measures given in Annexes B and C of IEC 61508-7.						
* Appropriate techniques/measures shall be selected according to the safety integrity level. Alternate or equivalent techniques/measures are indicated by a letter following the number. It is intended the only one of the alternate or equivalent techniques/measures should be satisfied. The choice of alternative technique should be justified in accordance with the properties, given in Annex C, desirable in the particular application.						

IEC 61508-7标准表格 C.4.5给出了何为“合适的编程语言”的具体描述定义：

- 编程语言应该是定义清晰且无异议的

.....

- 编程语言最好满足如下特性：

- 可分解为易于管理的小型软件模块
- 在软件模块之间限制数据的访问权限
- 限制定义变量的使用范围
- 其他各种可以限制错误发生的结构类型

在选择合适的C语言子集，遵循编程标准和使用静态代码分析工具的前提下，SIL所有的四个等级都高度推荐使用C编程语言。

Table C.1 – Recommendations for specific programming languages

9	C	R	–	NR	NR
10	C with subset and coding standard, and use of static analysis tools	HR	HR	HR	HR

NR: Not Recommended
HR: Highly Recommended

IEC 61508-7标准表格 C.4.2给出了编程语言子集的目标定义和相关描述:

目标

- 降低引入编程错误的可能性
- 增加发现潜在编程错误的可能性

描述

- 使用编程语言子集来排除一些容易犯错的结构类型
- 使用静态分析方法来发现编程语言的缺陷和错误

C-STAT: 静态代码分析

- 代码分析工具— 检测应用程序中的错误代码
 - 静态分析工具
 - 不执行目标代码的情况下分析代码
 - 运行时分析工具
 - 动态执行目标代码的过程中分析代码

Static analysis
applies *before*
compilation and
debug

```
int bounds_check (int i, int v)
{
    int a[5];
    a[i] = v;
    return a[i+1];
}

int divide (int i, int d)
{
    return i / d;
}

void access_memory (char* str)
{
    *p = (char) (sizeof str);
    if (*p)
    {
        p[100] = '1';
    }
    else
    {
        p[100] = 0;
    }
}
```

C / C++ code



Build chain
(compiler &
linker)

Runtime analysis applies
after compilation and
during debug/execution



Software debugger

- C-STAT是由IAR Systems自主开发的静态分析工具
 - 2015年2月首发
 - 支持C 和C++ 源码
- C-STAT是IAR Embedded Workbench的附加产品
 - 完全集成于EW
 - 无需额外安装程序
 - 不需要额外的license
 - 不支持第三方的编译和调试工具
- 软件类别和版本支持
 - IAR Embedded Workbench for ARM, from version 7.40以上
 - IAR Embedded Workbench for TI MSP430, from version 6.30以上
 - IAR Embedded Workbench for Atmel AVR32, from version 4.30以上
 - IAR Embedded Workbench for Renesas V850 4.20以上



- 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.



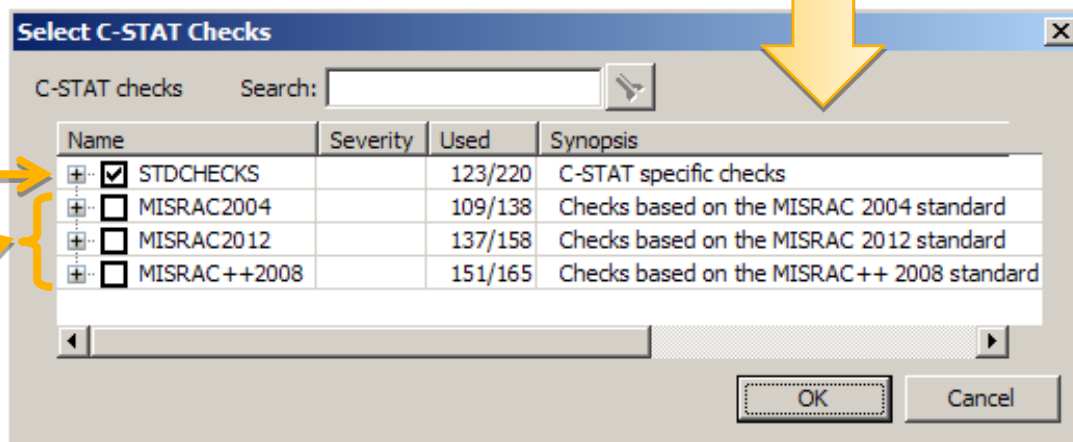
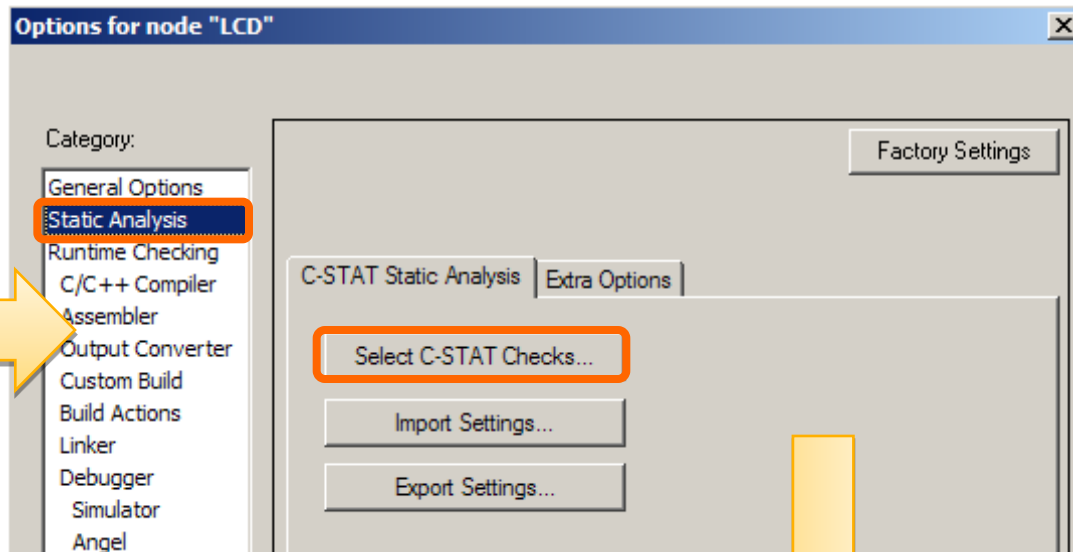
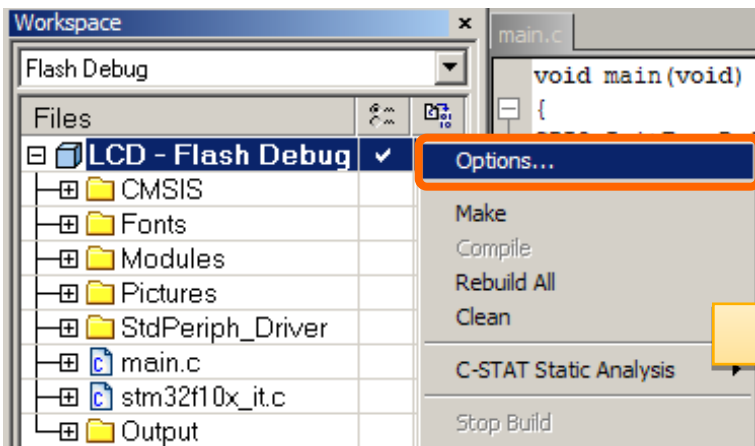
- Computer Emergency Response Team
- www.cert.org
- C/C++ secure coding standards, identifying insecure constructs which could expose a weakness or vulnerability in the software.
- Guidelines to avoid implementation, coding as well as low-level design errors.

- 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.

IAR EWARM C-STAT选项



CWE/CERT rules

MISRA C/C++ rules

C-STAT: 规则配置

Select C-STAT Checks

C-STAT checks Search:

Name	Severity	Used	Synopsis
<input type="checkbox"/> MISRAC2012		137/158	Checks based on the MISRAC 2012 standard
<input checked="" type="checkbox"/> MISRAC2012-Dir-4		2/6	Code design
<input checked="" type="checkbox"/> MISRAC2012-Dir-4.3	Low		Inline asm statements that are not encapsulated in functions
<input type="checkbox"/> MISRAC2012-Dir-4.4	Low		To allow comments to c
<input type="checkbox"/> MISRAC2012-Dir-4.6_a	Low		Uses of basic types cha
<input type="checkbox"/> MISRAC2012-Dir-4.6_b	Low		Typedefs of basic type
<input type="checkbox"/> MISRAC2012-Dir-4.9	Low		Function-like macros
<input checked="" type="checkbox"/> MISRAC2012-Dir-4.10	Low		Header files without #i
<input checked="" type="checkbox"/> MISRAC2012-Rule-1		All	A standard C environm
<input checked="" type="checkbox"/> MISRAC2012-Rule-2		4/5	Unused code
<input checked="" type="checkbox"/> MISRAC2012-Rule-3		All	Comments
<input checked="" type="checkbox"/> MISRAC2012-Rule-4		None	Character sets and lex
<input checked="" type="checkbox"/> MISRAC2012-Rule-5		All	Identifiers
<input checked="" type="checkbox"/> MISRAC2012-Rule-6		All	Types
<input checked="" type="checkbox"/> MISRAC2012-Rule-7		All	Literals and constants

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

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

IAR Embedded Workbench Help for target

Hide Locate Back Forward Home Print

Contents Index Search

Type in the keyword to find:

MISRAC2012-Dir-4.3

MISRAC2012-Dir-4.3

MISRAC2012-Dir-4.4

MISRAC2012-Dir-4.6_a

MISRAC2012-Dir-4.6_b

MISRAC2012-Dir-4.9

MISRAC2012-Rule-1.3_a

MISRAC2012-Rule-1.3_b

MISRAC2012-Rule-1.3_c

MISRAC2012-Rule-1.3_d

MISRAC2012-Rule-1.3_e

MISRAC2012-Rule-1.3_f

MISRAC2012-Rule-1.3_g

MISRAC2012-Rule-1.3_h

MISRAC2012-Rule-10.1_R2

MISRAC2012-Rule-10.1_R3

MISRAC2012-Rule-10.1_R4

MISRAC2012-Rule-10.1_R5

MISRAC2012-Rule-10.1_R6

MISRAC2012-Rule-10.1_R7

MISRAC2012-Rule-10.1_R8

MISRAC2012-Rule-10.2

MISRAC2012-Rule-10.3

MISRAC2012-Rule-10.4

MISRAC2012-Rule-10.6

MISRAC2012-Rule-10.7

MISRAC2012-Rule-10.8

MISRAC2012-Rule-11.1

MISRAC2012-Rule-11.3

MISRAC2012-Rule-11.4

MISRAC2012-Rule-11.7

MISRAC2012-Rule-11.8

MISRAC2012-Rule-11.9

MISRAC2012-Rule-12.1

Display

C-STAT checks : Descriptions of checks : MISRAC2012-Dir-4.3

MISRAC2012-Dir-4.3

Synopsis

Inline asm statements that are not encapsulated in functions

Enabled by default

Yes

Severity/Certainty

Low/Medium

Full description

(Required) Assembly language shall be encapsulated and isolated

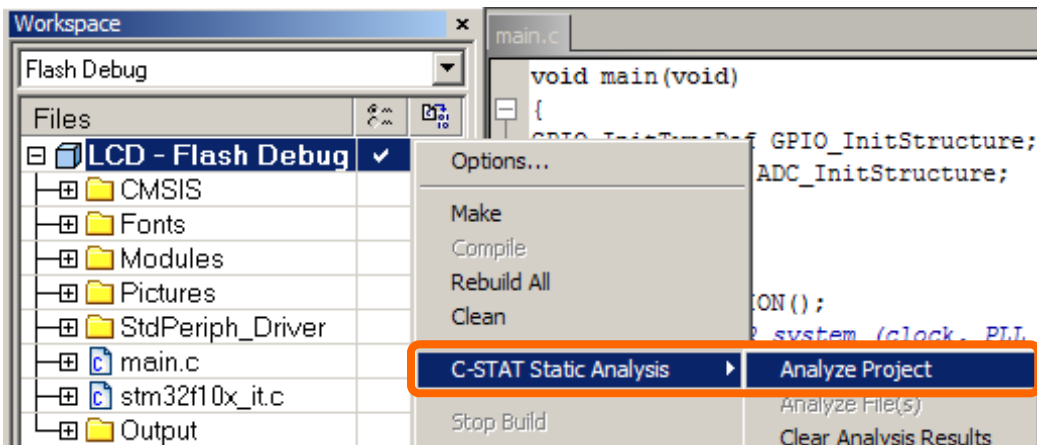
Coding standards

MISRA C:2012 Dir-4.3

(Required) Assembly language shall be encapsulated and isolated

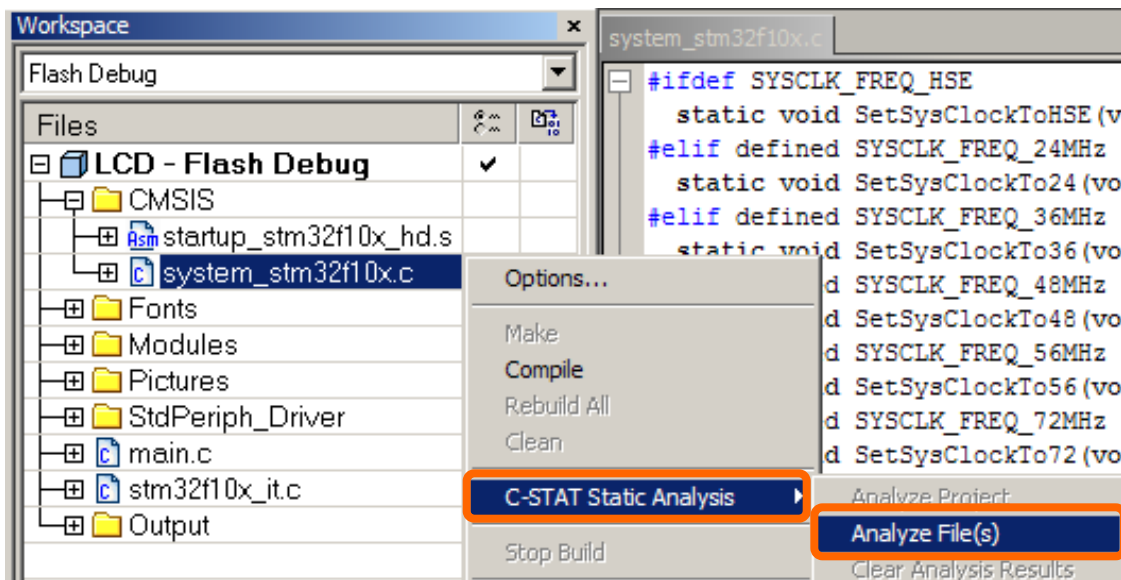
Code examples

C-STAT: 分析代码



Analyze the whole project

Analyze an individual source file or a group of source files



C-STAT: 分析结果

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

```
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 */
}
```

Message	Check	Severity	File
drv_glcd.c (19 messages)			drv_glcd.c
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

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

ATH-shift-bounds

Synopsis
Out of range shifts

Enabled by default
Yes

Severity/Certainty
Medium/Medium

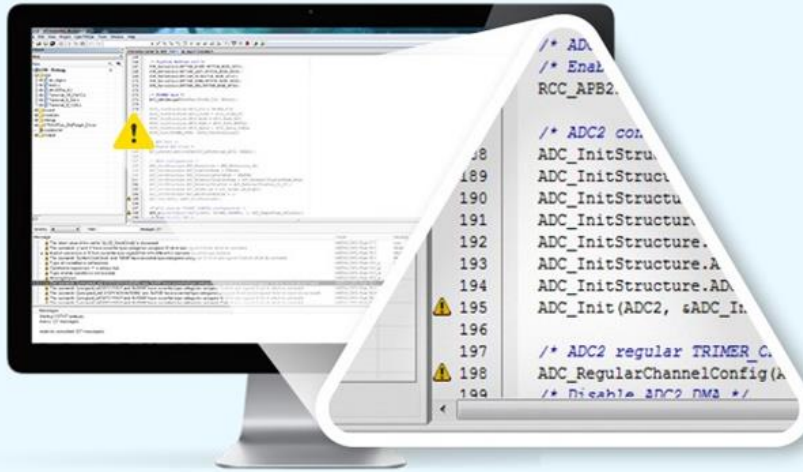
Full description
A shift operator on an n-bit argument may only shift between 0 and n-1 bits. In this case, the right-hand operand may be negative, or too large. This check is for all platforms. The behavior in this situation is undefined; the code may work as intended, or data could become erroneous.

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: 运行时分析工具

IAR C-STAT和IAR C-RUN

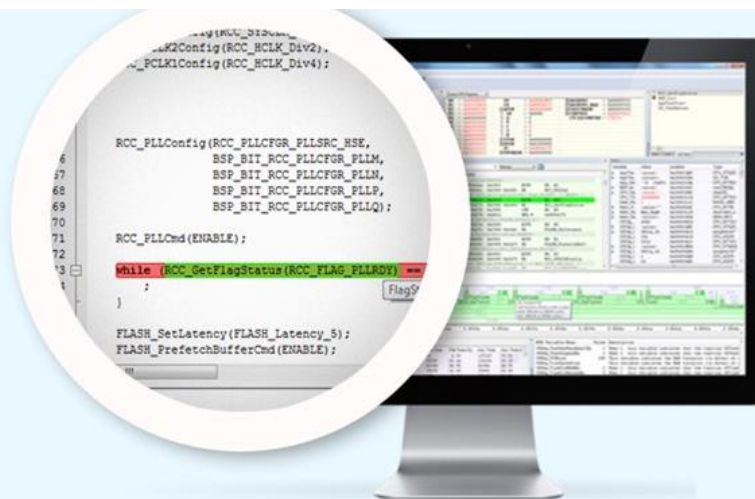


C-STAT Static analysis

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.



C-RUN: Runtime code analysis

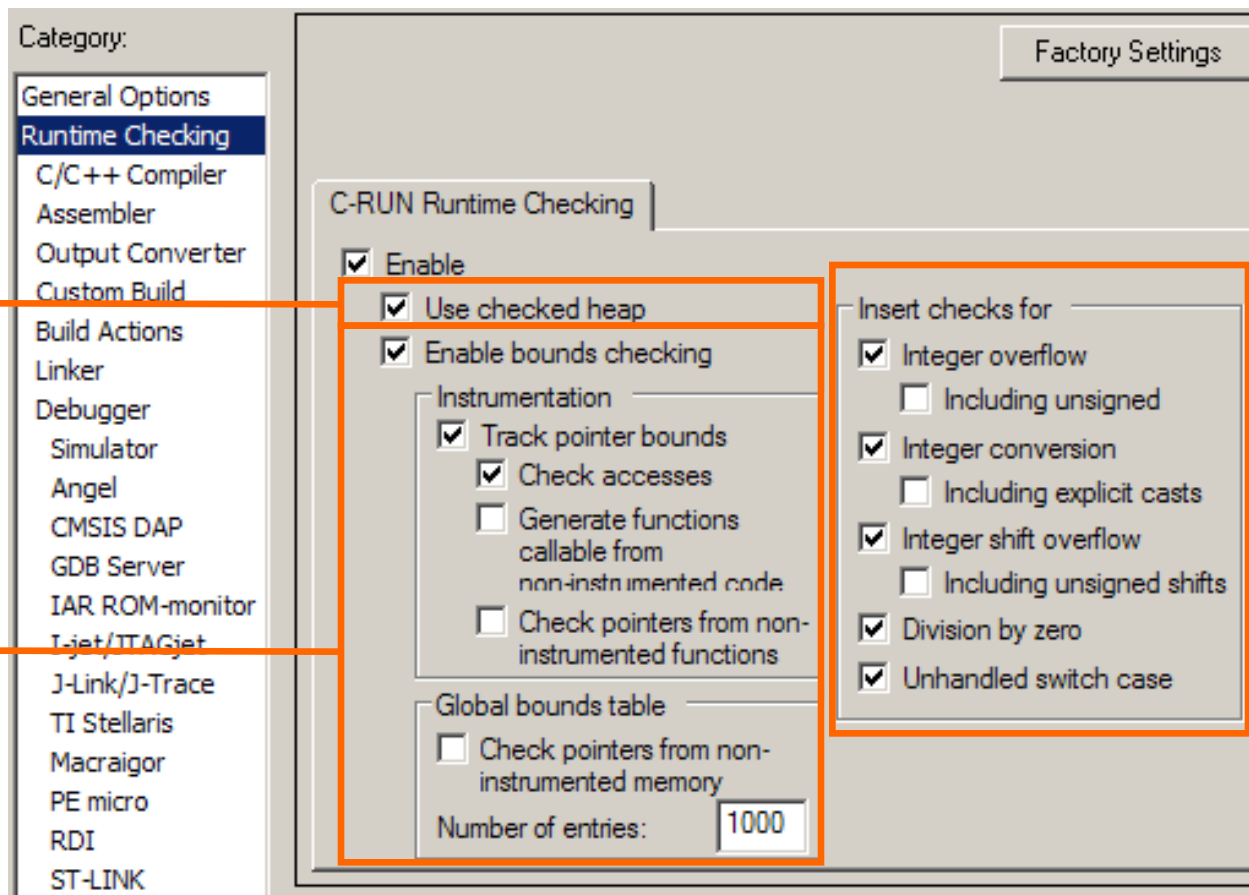
- C-RUN是由IAR Systems自主开发的动态分析工具
 - 2014年5月首发
 - 支持C 和C++ 源码
- C-RUN是IAR Embedded Workbench的附加产品
 - 完全集成于EW
 - 无需额外安装程序
 - 不需要额外的license
 - 不支持第三方的编译和调试工具
- 目标支持
 - IAR Embedded Workbench for ARM, from version 7.20以上
 - 支持所有的ARM内核

IAR EWARM C-RUN选项

Heap
checking

Bounds
checking

Arithmetic
checking



The screenshot shows the 'C-RUN Runtime Checking' settings in IAR EWARM. The 'Runtime Checking' category is selected in the left sidebar. The main panel has a 'Factory Settings' button and a 'C-RUN Runtime Checking' section. The 'Enable' checkbox is checked. Below it, 'Use checked heap' and 'Enable bounds checking' are checked. The 'Instrumentation' section has 'Track pointer bounds' checked, with 'Check accesses' also checked. 'Generate functions callable from non-instrumented code' and 'Check pointers from non-instrumented functions' are unchecked. The 'Global bounds table' section has 'Check pointers from non-instrumented memory' unchecked, and the 'Number of entries' is set to 1000. The 'Insert checks for' section has 'Integer overflow', 'Integer conversion', and 'Integer shift overflow' checked, with 'Including unsigned', 'Including explicit casts', and 'Including unsigned shifts' unchecked. 'Division by zero' and 'Unhandled switch case' are also checked.

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/ITAGjet
- J-Link/J-Trace
- TI Stellaris
- Macraigor
- PE micro
- RDI
- ST-LINK

Factory Settings

C-RUN Runtime Checking

- Enable
- 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
- Global bounds table
 - Check pointers from non-instrumented memory
 - Number of entries:
- Insert checks for
 - Integer overflow
 - Including unsigned
 - Integer conversion
 - Including explicit casts
 - Integer shift overflow
 - Including unsigned shifts
 - Division by zero
 - Unhandled switch case



检测整型溢出

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

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

- Insert checks for
- Integer overflow
 - Including unsigned
 - Integer conversion
 - Including explicit casts
 - Integer shift overflow
 - Including unsigned shifts
 - Division by zero
 - Unhandled switch case

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]		

检测整型转换

```
void main (void)
{
    int v1 = 0x8000;
    short v2;
    char v3;

    v2 = v1; /* 32-bit → 16-bit */
    v3 = v1; /* 32-bit → 8-bit */
}
```

Insert checks for

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

Default action: Stop Filter: Messages: 2

Messages	Source File	PC
Integer conversion failure	main.c 7:8-9	0x000000D2
Integer conversion failure	main.c 8:8-9	0x000000E4
Conversion changes the value from 32768 (0x000008000) to 0 (0x00).		
Call Stack		
main	main.c 8:3-10	
[_call_main + 0x9]		

检测移位溢出




```
void main (void)
{
    int i;
    short v1 = 1;
    int v2 = 1;

    for (i=0; i<32; i++)
    {
        v1 <<= 1; /* overflow: i>14 */
        v2 <<= 1; /* overflow: i>30 */
    }
}
```

Insert checks for

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

Default action: Stop Filter: Messages: 2

Messages	Source File	PC
 Shift overflow	main.c 9:15-22	0x0000010A
 Shift overflow	main.c 10:5-12	0x0000012A
Result is greater than the largest representable number: signed value 1073741824 (0x40000000) doubled 1 time(s).		
 Call Stack		
main	main.c 10:5-13	
[_call_main + 0x9]		

检测遗漏的switch-case语句


```
void main (void)
{
    int i;

    for (i=0; i<2; i++)
    {
        switch (i) /* case 1 is not handled */
        {
            case 0:
                break;
        }
    }
}
```

Insert checks for

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

Default action: Stop Filter: Messages: 1

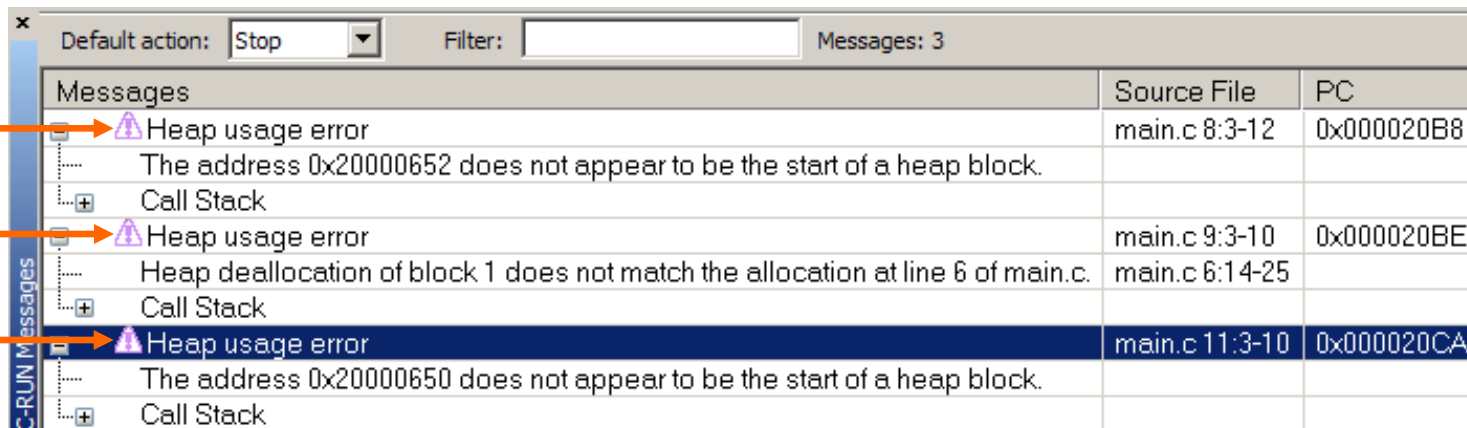
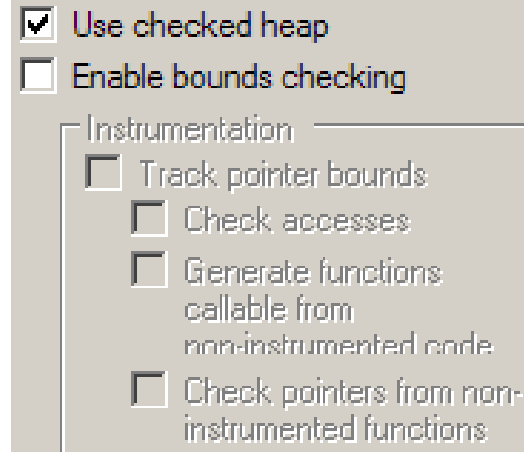
Messages	Source File	PC
 Unhandled case in switch	main.c 7:5-14	0x0000009C
Switch to undefined case label.		
Call Stack		
main	main.c 5:18-21	
[_call_main + 0x9]		

检测heap错误 - 1

```
#include <stdlib.h>

void main (void)
{
    char *c1 = (char *)malloc(10);
    char *c2 = new char[10];

    free(c1+2); /* not the start of a block */
    free(c2);   /* non-matched new and free */
    free(c1);
    free(c1);  /* free a block more than once */
}
```



Default action: Stop Filter: Messages: 3

Messages	Source File	PC
Heap usage error The address 0x20000652 does not appear to be the start of a heap block. Call Stack	main.c 8:3-12	0x000020B8
Heap usage error Heap deallocation of block 1 does not match the allocation at line 6 of main.c. Call Stack	main.c 9:3-10 main.c 6:14-25	0x000020BE
Heap usage error The address 0x20000650 does not appear to be the start of a heap block. Call Stack	main.c 11:3-10	0x000020CA

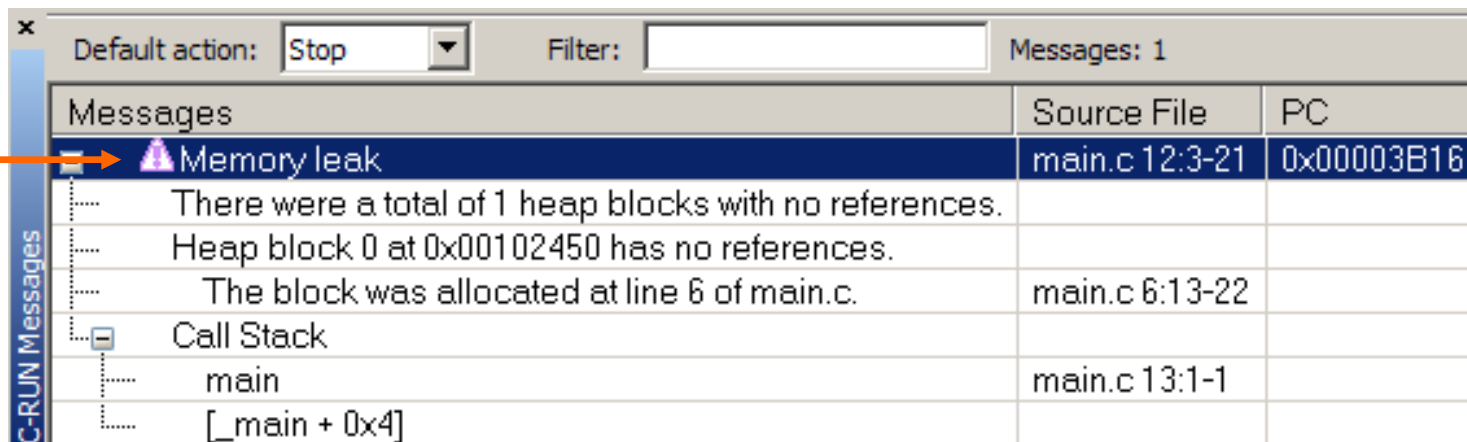
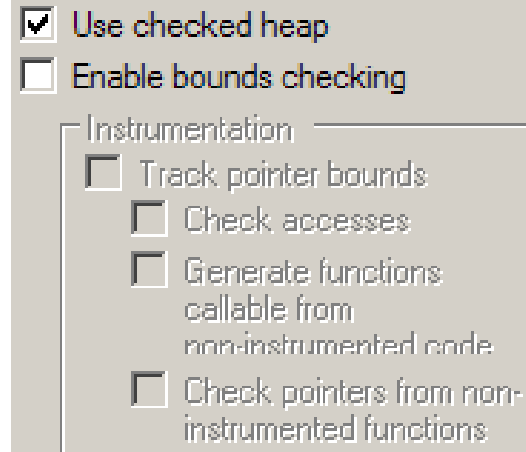
检测heap错误 - 2

```
#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();
}
```



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]	

检测越界访问

```
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

Enable bounds checking

Instrumentation


Track pointer bounds

Check accesses

Generate functions callable from non-instrumented code

Check pointers from non-instrumented functions

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		

优化你的开发流程

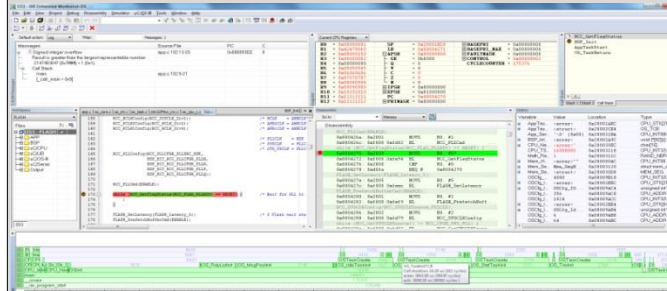
Implement your design in code

```
int bounds(int i, int v)
{
    int a[5];
    a[i] = v;
    return a;
}

int divide(int i, int v)
{
    return a[i] / v;
}

void access_memory(char* p)
{
    if (p[0])
        p[0x0c] = '1';
    else
        p[0x0c] = 0;
}
```

Build and debug the application



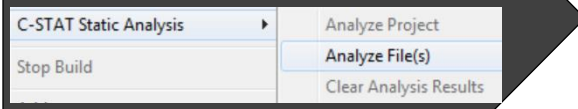
Release the application

```
10001 11010111010 01001
00011 11001100011 01110
10010 00101011110 01011
10111 10110011001 10011
00111 01010101101 11001

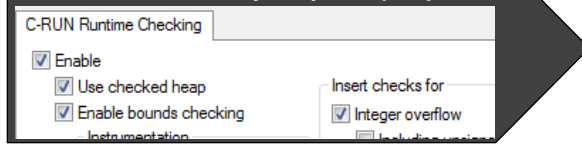
110010101110011
101010110011001
```



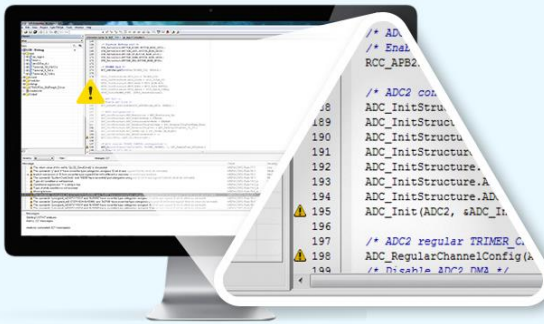
Let C-STAT analyze your code



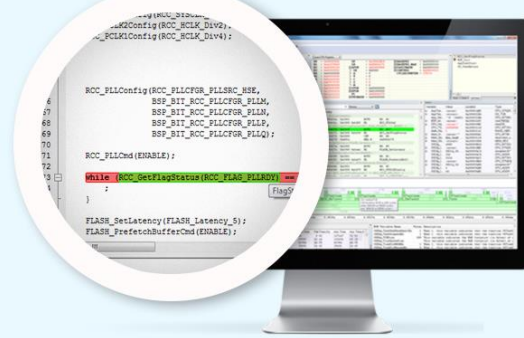
Let C-RUN analyze your project



Review potential issues



Investigate runtime errors



Requirements

Design

Implementation

Verification

Maintenance

IAR Systems: 值得信赖的战略伙伴

- 多种架构，一站式解决方案
- 高效高性能代码
- RTOS & Middleware集成
- 高级跟踪调试
- 功耗调试
- C-STAT静态代码分析
- C-RUN运行时代码分析
- 堆栈分析和跟踪
- 功能安全认证
- 全球专业技术支持

