Keeping Safe at C

IAR Systems, Shanghai
ryan.sheng@iar.com

IAR SYSTEMS

Agenda

- Introduction of IAR Systems
- Functional Safety Certificate
- C: Safe or Not
- C-STAT: Static Code Analysis
- C-RUN: Runtime Code Analysis

Introduction of IAR Systems

IAR SYSTEMS

IAR Systems

- Established in 1983
- Headquarter: Uppsala, Sweden
- 170+ employees
- Support for 10,000+ devices
    - 3000+ ARM devices
- A world-leading embedded development tools vendor
- Main products
    - IAR Embedded Workbench: C/C++ Compiler & Debugger Tools
    - IAR visualSTATE: State-Machine Modeling & Software Design Tools
    - IAR I-jet / I-scope / JTAGjet: Debugging & Trace Probes
- China office
    - Shanghai, 021-63758658

Strategic collaboration with Freescale

IAR SYSTEMS

- Long partnership with Freescale
- Initiated close cooperation around HC12 & S12
- EWCF: released on 2007
- EWS08: released on 2008
- EWARM is the most widely used commercial tool chain for ARM-based processors
- Expand the Freescale ecosystem
- IAR Embedded Workbench
  - EWHCS12: HC12 & S12 MCU
  - EWCF: ColdFire & ColdFire+ MCU/MPU
  - EWS08: S08 MCU
  - EWARM: Kinetis, i.MX, Vybrid, MC1322x, …

K24
K30
K40
K50
K60
K70
KE02
KE04
KE06
KE14
KE15
KEAZ
KL02
KL03
KL04
KL05
KL14
KL15
KL16
KL17
KL24
KL25
KL26
KL27
KL33
KL34
KL36
KL43
KL46
KM13
KM14
KM32
KM33
KM34
KM38
KV10
KV30
KV31
KV40
KV43
KV44
KV45
KV46
KW01

IAR Embedded Workbench for ARM

IAR SYSTEMS

IDE

eclipse READY

Editors

Source code control systems

SUBVERSION

PLUGINS

IAR Embedded Workbench IDE

IDE tools

- Editor
- Project manager
- Library builder
- Librarian

Build tools

- IAR C/C++ Compiler
- Assember
- Linker

IAR C-SPY Debugger

- Simulator driver
- Hardware system drivers
- Power debugging
- RTOS plug-ins

µC/Probe

Micriµm

expresslogic

SEGGER freeRTOS eForce

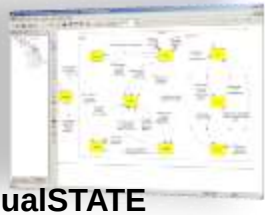
Micro Digital freescale Quadros

SCIOPTA CMX SYSTEMS MITTENSTEIN

Configuration tools

infineon DAVE™

freescale Processor Expert Software



visualSTATE

EWARM: Product variants

- 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

Functional Safety Certificate

IAR SYSTEMS

What is Functional Safety?

- Simply put, it means that the overall safety of an embedded system depends on the equipment operating correctly in response to its inputs.
  - This includes erroneous inputs
  - Also includes hardware failures
- Traditionally, only safety-critical industries have been interested in functional safety.
  - Automotive
  - Avionics
  - Medical
- However, other industries are also seeing the benefit.

Functional Safety Standards

The IEC 61508 Standard

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.

Certified Tools from IAR Systems

Functional safety standards

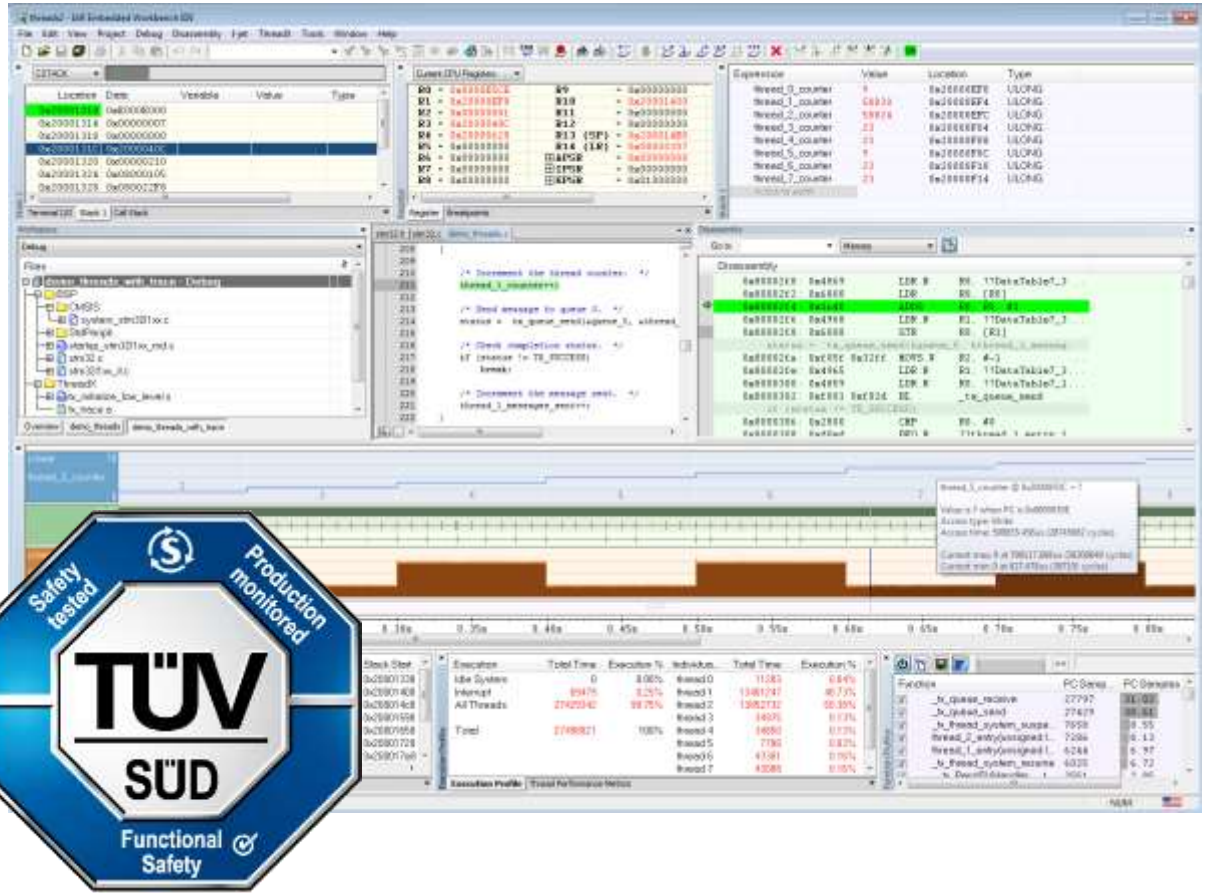
- IEC 61508
- ISO 26262
- EN 50128

Simplified validation

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

Guaranteed support and upgrade

- Cover the product life cycle
- Prioritized technical support
- Validated service packs
- Report of known problems



www.iar.com/safety

EWARMFS

- EWARMFS
  - The Functional Safety edition of IAR Embedded Workbench for ARM
- Certified by TÜV SÜD
- Functional safety standards
  - IEC 61508-3:2010 (SIL 3)
    For electrical, electronic and programmable systems in all kinds of industry.
  - ISO 26262-8:2011 (ASIL D)
    Safety standard for road vehicles, derived from IEC 61508.
  - EN 50128
    Safety standard for railway control and protection systems.

C: Safe or Not

IAR SYSTEMS

The IEC 61508 Standard

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.

Suitable programming language

Table C.4.5 of IEC 61508-7 gives a general description of "suitable programming language":

- The language should be fully and unambiguously defined.

……

- The language should encourage:
  - *The use of small and manageable software modules;*
  - *Restriction of access to data in specific software modules;*
  - *Definition of variable subranges; and*
  - *Any other type of error-limiting constructs.*

Recommendation of languages

With the right C subset, coding standard and the use of static analysis tools, C can be a Highly Recommended programming language for all 4 levels of SIL.

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

Language with Subset

Table C.4.2 of IEC 61508-7 gives the aim and description of language subsets:

Aim:

- To reduce the probability of introducing programming faults;
- Increase the probability of detecting any remaining faults.

Description:

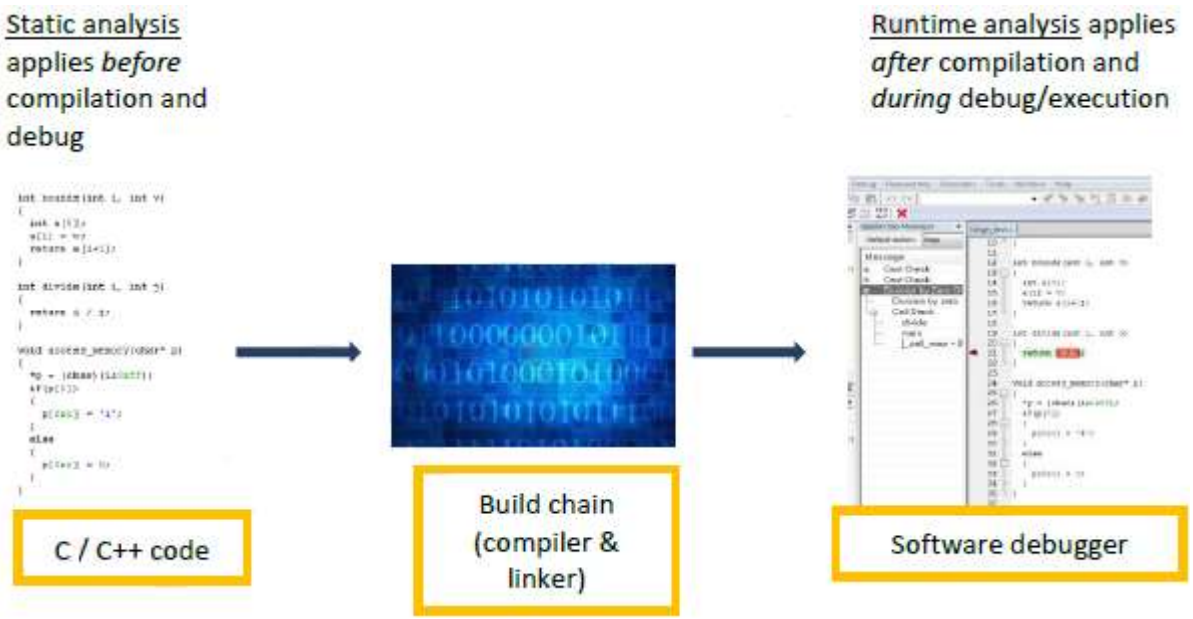
- The language is examined to determine programming constructs which are either error-prone or difficult to analyze, for example, using static analysis methods.
- A language subset is then defined which excludes these constructs.

C-STAT: Static Code Analysis

IAR SYSTEMS

Code analysis tools

- Code analysis tools – Detecting erroneous code in the application
  - Static analysis tools
    - Analyze the source code without executing them
  - Runtime analysis tools
    - Analyze the source code dynamically during execution



C-STAT: Static code analysis

- C-STAT is a static analysis tool developed by IAR Systems
  - Launched in Feb, 2015
  - Both C and C++ source code are supported
- C-STAT is an add-on product of IAR Embedded Workbench
  - Fully integrated
  - No additional installation
  - No separate license
  - Cannot work with 3rd-party compiler & debugger tools
- Target support
  - IAR Embedded Workbench for ARM, from version 7.40
  - IAR Embedded Workbench for TI MSP430, from version 6.30
  - IAR Embedded Workbench for Atmel AVR32, from version 4.30

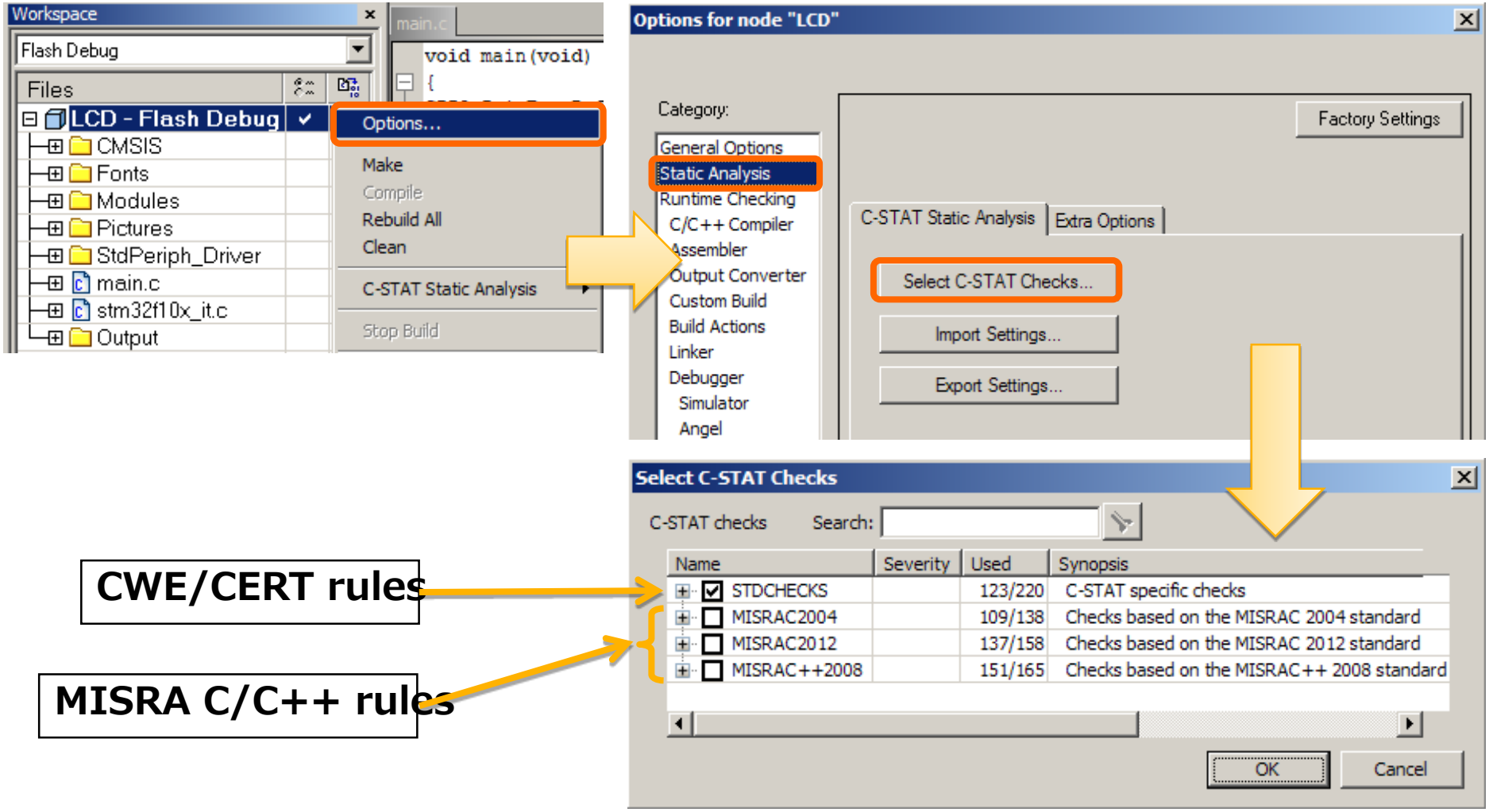
C-STAT: What does it check

- 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.

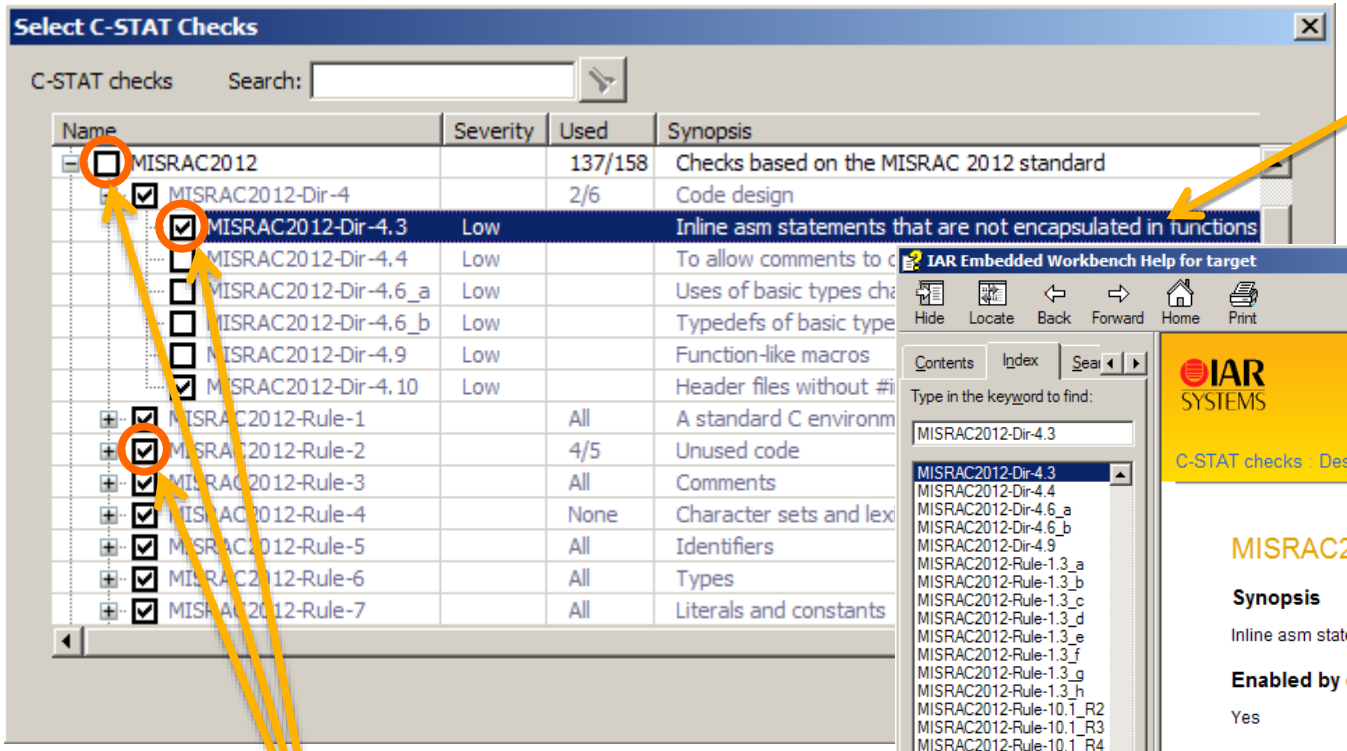
- Computer Emergency Response Team
- www.cert.org
- C/C++ secure coding standards, identifying insecure constructs which could expose a weakness or vulnerability in the software.
- Guidelines to avoid implementation, coding as well as low-level design errors.

- 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.

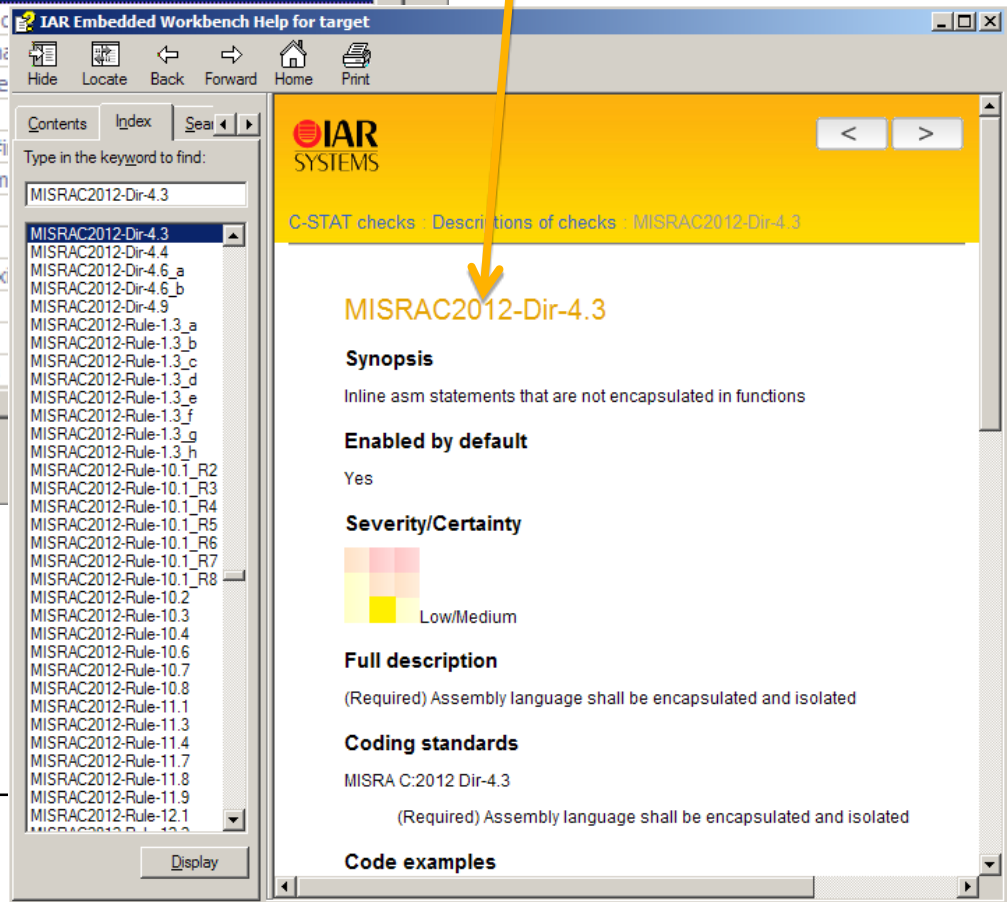
C-STAT options in IAR EWARM



C-STAT: Rules configuration

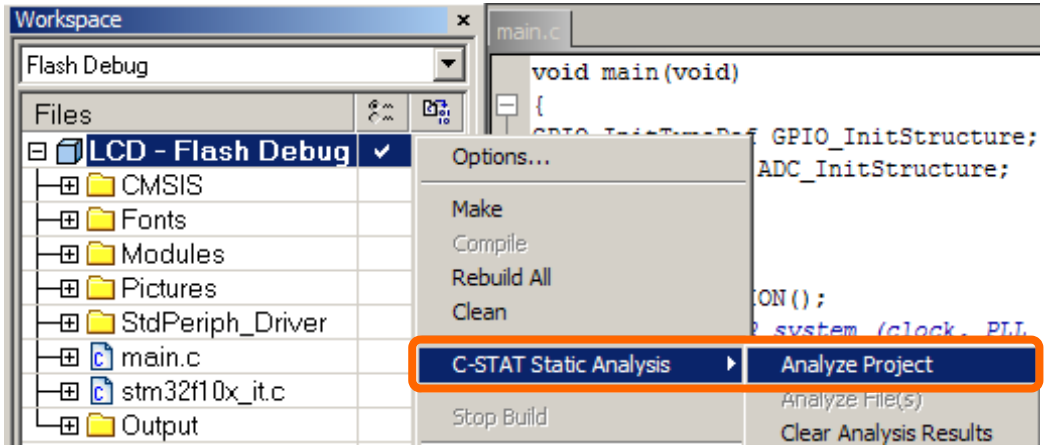


Highlight a rule and press F1 to show the detailed description.

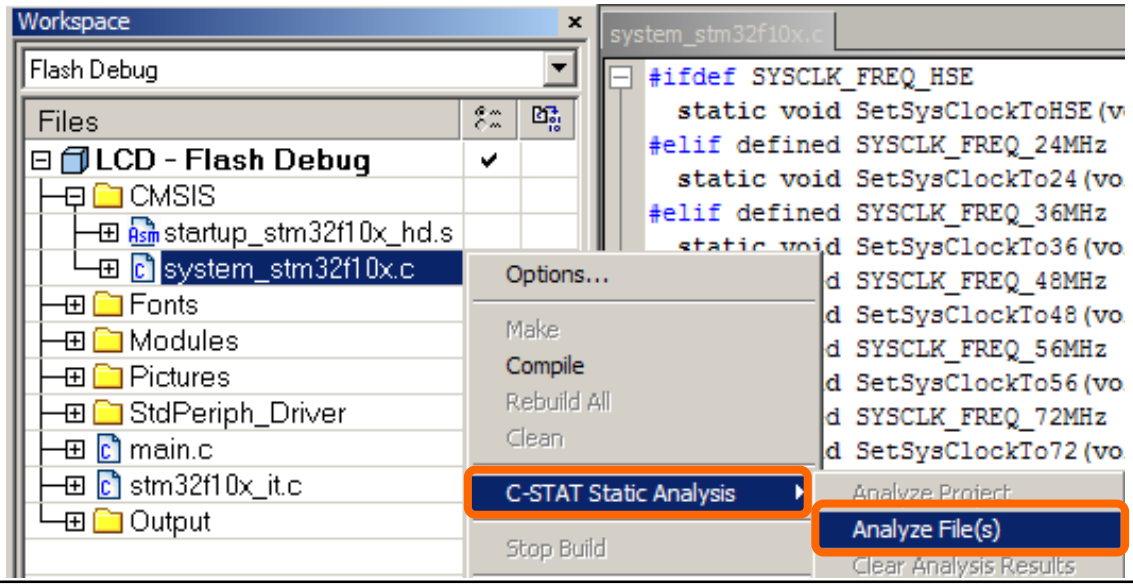


Enable or disable a set of rules or any individual rule.

C-STAT: Analyze the code



Analyze the whole project



Analyze an individual source file or a group of source files

C-STAT: Result of analysis

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

Contents | Index | Search

ATH-shift-bounds

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: Runtime Code Analysis

IAR SYSTEMS

IAR C-STAT and 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 is a runtime analysis tool developed by IAR Systems
  - Launched in May, 2014
  - Both C and C++ source code are supported
- C-RUN is an add-on product of IAR Embedded Workbench
  - Fully integrated
  - No additional installation
  - No separate license
  - Cannot work with 3rd-party compiler & debugger tools
- Target support
  - IAR Embedded Workbench for ARM, from version 7.20
  - All ARM cores are supported

C-RUN options in IAR EWARM

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

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 integer conversion

```
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
- [x] Integer conversion
  - [x] 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
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]		

Detecting shift overflow

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

C-RUN Messages

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]		

Detecting unhandled switch-case

```
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
- [x] Unhandled switch case

C-RUN Messages

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]		

Detecting heap errors - 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 */
}
```

- [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: 3

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

Detecting heap errors - 2

```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

```c
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		

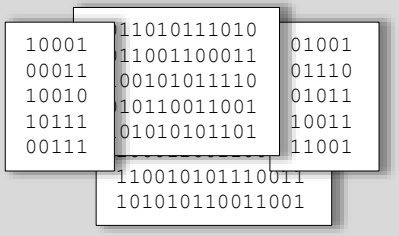
Take full control of your development

IAR SYSTEMS

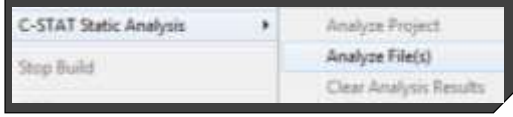
Implement your design in code

Build and debug the application

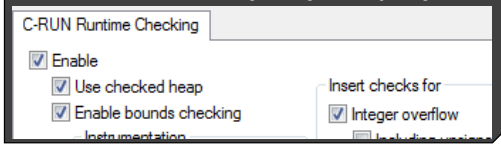
Release the application



Let C-STAT analyze your code



Let C-RUN analyze your project



Review potential issues

Investigate runtime errors

Requirements → Design → Implementation → Verification → Maintenance

IAR Systems: Your strategic partner

- Different architecture, one solution
- Most efficient & high performance code
- Freescale MQX™ RTOS integration
- Freescale Processor Expert integration
- Advanced trace debugging
- Power debugging
- C-STAT static code analysis
- C-RUN runtime code analysis
- Stack usage analysis & tracking
- Functional safety certificate
- Global professional technical support