TASKING®

TASKING TriCore v6.2r2 Inspector User Guide

MA160-741 (v1.0r4) July 13, 2023

Table of Contents

Manual Purpose and Structure

Manual Purpose

You should read this manual if you want to know:

- how to use the TASKING TriCore v6.2r2 Inspector
- the features of the TASKING TriCore v6.2r2 Inspector

Manual Structure

Chapter 1, *Installing the Software*

Explains how to install and license the TASKING TriCore v6.2r2 Inspector.

Chapter 2, *Introduction to the TASKING Inspector*

Contains an introduction to the TASKING TriCore v6.2r2 Inspector and contains an overview of the features.

Chapter 3, *Using the Inspector*

Explains how to use the TASKING TriCore v6.2r2 Inspector.

Chapter 4, *Tool Options*

Contains an overview of all the Inspector specific options of the tools.

Related Publications

- Getting Started with the TASKING VX-toolset for TriCore
- TASKING VX-toolset for TriCore User Guide
- TriCore 1 32-bit Unified Processor Core, Volume 1 Core Architecture, V1.3 & V1.3.1 Architecture User's Manual, V1.3.8 [2007-11, Infineon]
- TriCore 1 32-bit Unified Processor Core, Volume 2 Instruction Set, V1.3 & V1.3.1 Architecture User's Manual, V1.3.8 [2007-11, Infineon]
- TC1xxx User's Manual, V2.0 [2007, Infineon]
- TriCore 1 32-bit Unified Processor Core, Embedded Applications Binary Interface (EABI), V1.3, V1.3.1 & V1.6 Architecture User's Manual, v2.5 [2008-01, Infineon]
- AURIX™ TC21x/TC22x/TC23x Family User's Manual, V1.1 [2014-12, Infineon]
- AURIX™ TC26x A-Step User's Manual, V1.1 [2013-12, Infineon]
- AURIX™ TC26x B-Step User's Manual, V1.2 [2014-02, Infineon]

- AURIX™ TC27x User's Manual, V1.4 [2013-11, Infineon]
- AURIX™ TC27x B-Step User's Manual, V1.4.1 [2014-02, Infineon]
- AURIX™ TC27x C-Step User's Manual, V2.2 [2014-12, Infineon]
- AURIX™ TC27x D-Step User's Manual, V2.2 [2014-12, Infineon]
- AURIX™ TC29x A-Step User's Manual, V1.1.1 [2014-01, Infineon]
- AURIX™ TC29x B-Step User's Manual, V1.3 [2014-12, Infineon]
- AURIX™ TC3xx Target Specification, V2.5.1 [2018-04, Infineon]
- AURIX™ TC3xx User's Manual, V2.0.0 [2021-02, Infineon]
- AURIX™ TC35x User's Manual Appendix, V1.6.0 [2020-08, Infineon]
- AURIX™ TC37x User's Manual Appendix, V1.6.0 [2020-08, Infineon]
- AURIX™ TC38x User's Manual Appendix, V1.6.0 [2020-08, Infineon]
- AURIX™ TC39x-B User's Manual Appendix, V1.6.0 [2020-08, Infineon]

Chapter 1. Installing the Software

This chapter guides you through the installation process of the TASKING® TriCore v6.2r2 Inspector. It also describes how to license the software.

In this manual, **TASKING TriCore v6.2r2 Inspector** and **Inspector** are used as synonyms.

1.1. Installation for Windows

System Requirements

Before installing, make sure the following minimum system requirements are met:

- Windows 7 or higher
- 2 GHz Pentium class processor
- 4 GB memory
- 500 MB free hard disk space

Installation

1. If you received a download link, download the software and extract its contents.

   - or -

   If you received an USB flash drive, insert it into a free USB port on your computer.

2. Run the installation program (**setup.exe**).

   The TASKING Setup dialog box appears.

3. Select a product and click on the **Install** button. If there is only one product, you can directly click on the **Install** button.

4. Follow the instructions that appear on your screen. During the installation you need to enter a license key, this is described in Section 1.2, *Licensing*.

1.2. Licensing

TASKING products are protected with TASKING license management software (TLM). To use a TASKING product, you must install that product and install a license.

The following license types can be ordered from TASKING.

Node-locked license

A node-locked license locks the software to one specific computer so you can use the product on that particular computer only.

For information about installing a node-locked license see Section 1.2.3.2, *Installing Server Based Licenses (Floating or Node-Locked)* and Section 1.2.3.3, *Installing Client Based Licenses (Node-Locked)*.

Floating license

A floating license is a license located on a license server and can be used by multiple users on the network. Floating licenses allow you to share licenses among a group of users up to the number of users (seats) specified in the license.

For example, suppose 50 developers may use a client but only ten clients are running at any given time. In this scenario, you only require a ten seats floating license. When all ten licenses are in use, no other client instance can be used. Also a linger time is in place. This means that a license seat is locked for a period of time after a user has stopped using a client. The license seat is available again for other users when the linger time has finished.

For information about installing a floating license see Section 1.2.3.2, *Installing Server Based Licenses (Floating or Node-Locked)*.

License service types

The license service type specifies the process used to validate the license. The following types are possible:

- **Client based** (also known as 'standalone'). The license is serviced by the client. All information necessary to service the license is available on the computer that executes the TASKING product. This license service type is available for node-locked licenses only.
- **Server based** (also known as 'network based'). The license is serviced by a separate license server program that runs either on your companies' network or runs in the cloud. This license service type is available for both node-locked licenses and floating licenses.

  Licenses can be serviced by a cloud based license server called "**TASKING Remote License Server**". This is a license server that is operated by TASKING. Alternatively, you can install a license server program on your local network. Such a server is called a "**TASKING Local License Server**". You have to configure such a license server yourself. The installation of a TASKING local license server is not part of this manual. You can order it as a separate product (SW000089).

  The benefit of using the TASKING Remote License Server is that product installation and configuration is simplified.

  Unless you have an IT department that is proficient with the setup and configuration of licensing systems we recommend to use the facilities offered by the TASKING Remote License Server.

1.2.1. Obtaining a License

You need a license key when you install a TASKING product on a computer. If you have not received such a license key follow the steps below to obtain one. Otherwise, you cannot install the software.

Obtaining a server based license (floating or node-locked)

- Order a TASKING product from TASKING or one of its distributors.

  A license key will be sent to you by email or on paper.

If your node-locked server based license is not yet bound to a specific computer ID, the license server binds the license to the computer that first uses the license.

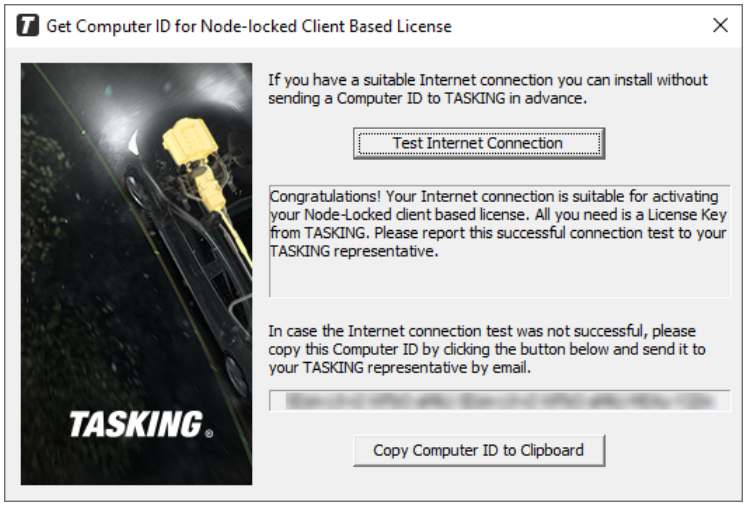
Obtaining a client based license (node-locked)

To use a TASKING product on one particular computer with a license file, TASKING needs to know the computer ID that uniquely identifies your computer. You can do this with the **getcid** program that is available on the TASKING website. The detailed steps are explained below.

1. Download the **getcid** program from https://www.tasking.com/support/tlm/downloads.

2. Execute the **getcid** program on the computer on which you want to use a TASKING product. The tool has no options. For example,

   `getcid_`*`version`*

   The computer ID is displayed in the lower part of the dialog.

3. Order a TASKING product from TASKING or one of its distributors and supply the computer ID.

   A license key and a license file will be sent to you by email or on paper.

When you have received your TASKING product, you are now ready to install it.

1.2.2. Frequently Asked Questions (FAQ)

If you have questions or encounter problems you can check the support page on the TASKING website.

https://www.tasking.com/support/tlm/faqs

This page contains answers to questions for the TASKING license management system TLM.

If your question is not there, please contact your nearest TASKING Sales & Support Center or Value Added Reseller.

1.2.3. Installing a License

The license setup procedure is done by the installation program.

If the installation program can access the internet then you only need the licence key. Given the license key the installation program retrieves all required information from the remote TASKING license server. The install program sends the license key and the computer ID of the computer on which the installation program is running to the remote TASKING license server, no other data is transmitted.

If the installation program cannot access the internet the installation program asks you to enter the required information by hand. If you install a node-locked client based license you should have the license file at hand (see Section 1.2.1, *Obtaining a License*).

Floating licenses are always server based and node-locked licenses can be server based. All server based licenses are installed using the same procedure.

1.2.3.1. Configure the Firewall in your Network

For using the TASKING license servers the TASKING license manager tries to connect to the remote license servers `lic1.tasking.com`, `lic2.tasking.com`, `lic3.tasking.com`, `lic4.tasking.com` at the TCP ports 8080, 8936 or 80. Make sure that the firewall in your network is transparently enabled for one of these ports.

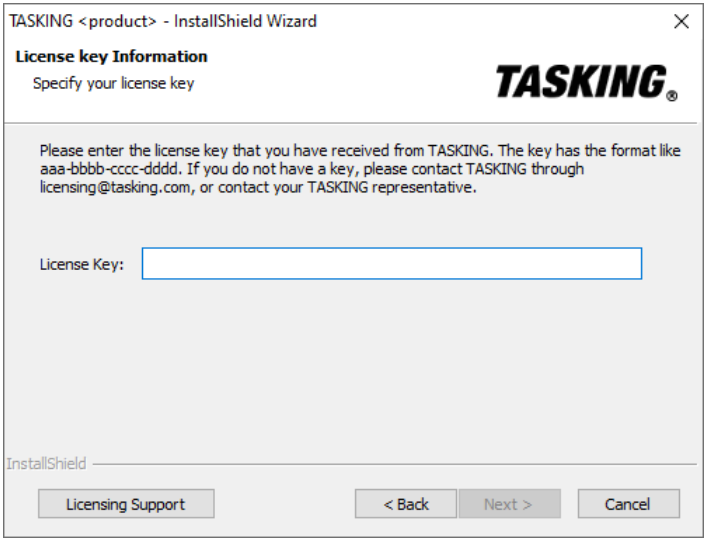
1.2.3.2. Installing Server Based Licenses (Floating or Node-Locked)

If you do not have received your license key, read Section 1.2.1, *Obtaining a License* before you continue.

1. If you want to use a local license server, first install and run the local license server before you continue with step 2. You can order a local license server as a separate product (SW000089).

2. Install the TASKING product and follow the instructions that appear on your screen.

   The installation program asks you to enter the license information.



3. In the **License Key** field enter the license key you have received from TASKING and click **Next** to continue.

   *The installation program tries to retrieve the license information from a remote license server. Wait until the license information is retrieved. If the license information is retrieved successfully subsequent dialogs are already filled-in and you only have to confirm the contents of the dialogs by clicking the **Next** button. If the license information is not retrieved successfully you have to enter the information by hand.*

4. Select your **License Type** and click **Next** to continue. If the license type is already filled in and grayed out, you can just click **Next** to continue.

   You can find the license type in the email or paper that contains the license key.

5. (For floating licenses only) Select **Remote license server** to use one of the remote license servers, or select **Local license server** for a local license server. The latter requires optional software.

   (For local license server only) specify the **Server name** and **Server port** of the local license server.

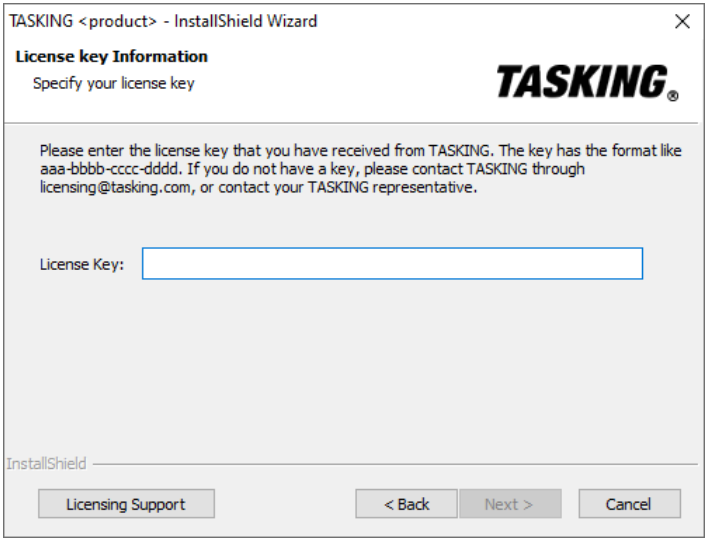
6. Click **Next** and follow the rest of the instructions to complete the installation.

1.2.3.3. Installing Client Based Licenses (Node-Locked)

If you do not have received your license key and license file, read Section 1.2.1, *Obtaining a License* before continuing.

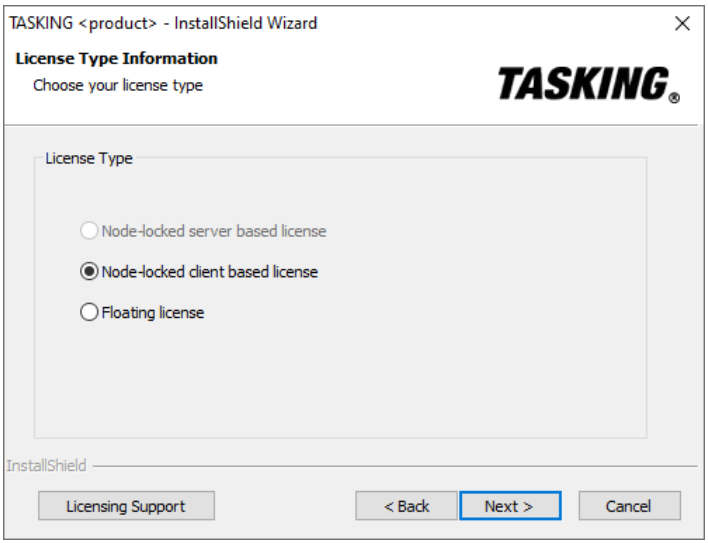
1. Install the TASKING product and follow the instructions that appear on your screen.

   The installation program asks you to enter the license information.

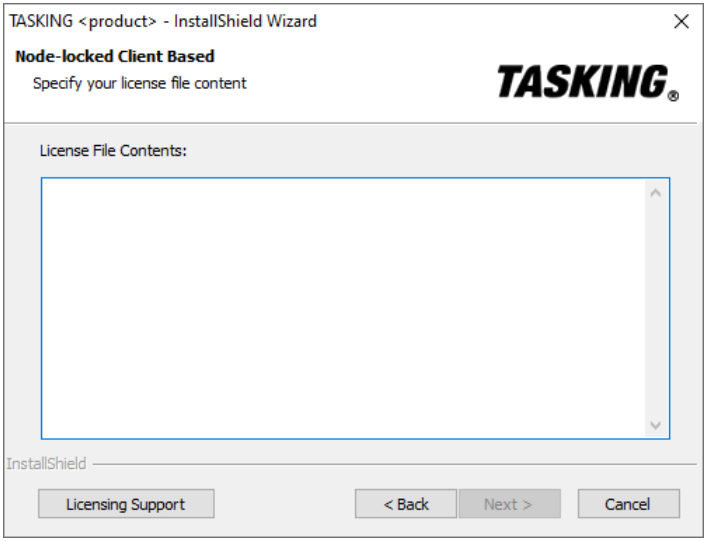


2. In the **License Key** field enter the license key you have received from TASKING and click **Next** to continue.

   *The installation program tries to retrieve the license information from a remote license server. Wait until the license information is retrieved. If the license information is retrieved successfully subsequent dialogs are already filled-in and you only have to confirm the contents of the dialogs by clicking the **Next** button. If the license information is not retrieved successfully you have to enter the information by hand.*



3. Select **Node-locked client based license** and click **Next** to continue.



4. In the **License File Contents** field enter the contents of the license file you have received from TASKING.

   The license data is stored in the file licfile.txt in the etc directory of the product (<install_dir>\etc).

5. Click **Next** and follow the rest of the instructions to complete the installation.

Chapter 2. Introduction to the TASKING Inspector

The TASKING TriCore v6.2r2 Inspector is a product which allows you to ascertain whether compilation of your code is affected by known issues in various components of the TASKING VX-toolset for TriCore v6.2r2 as presented on the TASKING Issues Portal. Essentially, this TASKING TriCore v6.2r2 Inspector v1.0r4 is a copy of the TriCore v6.2r2p2 toolset that does not produce any executable code.

The TASKING Inspector is a useful tool to determine if you need a patch for the TASKING VX-toolset for TriCore v6.2r2 or not. Depending on the status of your project a patch can be applied easily or an extensive retest and re-qualification of the complete project is required. Without an Inspector tool it is necessary to go trough the complete project source code by hand or write complex search scripts. Even with such complex scripts it is not possible to determine correctly if the code is impacted or not. The Inspector tool is setup in a way that it does not require any complex search scripts or hand review of the customer software. Only when an issue is detected you need to review the affected source code and/or generated assembly code. Defined by the usage of the Inspector it is guaranteed that each software component for a particular project is checked and the probability of failures is reduced a lot.

Issue detectors

Minimally required snippets of code called 'detectors' are inserted into the original tools, which can detect the fact that the currently compiled code will be affected by a certain known issue and then the detectors report that fact.

Some detectors cannot determine such fact with 100% certainty. Such detectors report that your code is potentially affected by the certain known issue and it's up to you to verify that.

By default all existing detectors are enabled within the Inspector and there are options that allow you to enable and disable particular detectors.

Features of the TASKING Inspector

- Detect known issues in your source code that definitely impact the tool output (no false positives)
- Detect known issues in your source code that potentially impact the tool output (false positive is possible)
- Clarify detection of a certain issue by comparing the assembly source files affected and not affected by that issue after its potential occurrence has been detected.
- Produce diagnostic messages

TASKING Inspector v1.0r4 use

You can use the TASKING Inspector v1.0r4 for the TASKING VX-toolset for TriCore v6.2r2 product and any of the patches, with the following restrictions:

- When you use the Inspector with the TriCore v6.2r2 product or with v6.2r2p1 applied, the Inspector does not detect issues fixed in TriCore v6.2r2p2.

- The level of confidence in the error detection of the Inspector is high if your software is compiled with the TASKING VX-toolset for TriCore v6.2r2p2 and degrades if other patch versions of the compiler are used.

2.1. Product Overview

The TASKING Inspector has all the same command-line tools as the TASKING VX-toolset for TriCore v6.2r2. The Eclipse IDE and debugger are not part of the TASKING Inspector. The linker cannot produce any executable code.

The following tools have Inspector capabilities. They are equivalent to the original tools without the "insp_" prefix, and contain additional options.

- Control Program: **insp_cctc**
- C++ Compiler: **insp_cptc**
- C Compiler: **insp_ctc**
- Assembler: **insp_astc**
- Linker: **insp_ltc**

For information about the individual tools, see the corresponding tools in the *TASKING VX-toolset for TriCore v6.2r2 User Guide*.

This manual only describes the additional functionality and options.

Chapter 3. Using the Inspector

The TASKING TriCore v6.2r2 Inspector is best used in conjunction with the TASKING VX-toolset for TriCore v6.2r2.

The Inspector accepts the same input as the compiler toolset and issues diagnostic messages about constructs in the input that can cause the compiler to malfunction. The components of the Inspector have the same structure as the components of the compiler toolset, and are identified with the prefix **insp_** in their naming convention.

You should use the Inspector in addition to the verification requirements specified by the FuSa standard that is applicable to your use case to obtain a high level of confidence that the behavior of your software is not affected by a known compiler issue.

This is achieved by:

- Analyzing the diagnostics provided by the Inspector in conjunction with the source code that causes the diagnostic message.
- Applying corrective actions to assure that your software or system satisfies its requirements, which may include:
  - Upgrade your compiler to a higher patch level that does not contain the identified malfunction.
  - Apply the mitigations that are provided on the TASKING Issues Portal to fix or workaround an identified malfunction.
  - Take other measures to assure that the detected issues do not impact the intended function of your software.

3.1. Detectors for Known Issues

Depending on the tools the Inspector can detect several known issues. For the list of issues you can detect, see the release notes of the TASKING TriCore v6.2r2 Inspector. For details about the issue, you can inspect the corresponding issues on the TASKING Issues Portal https://issues.tasking.com/?project=TCVX&version=v6.2r2p2. In the overview list you can see in the Inspector column which issues are covered by an Inspector. In the detailed issue description, the Inspector field shows which Inspector version detects the issue.

3.2. Detecting Issues with the Inspector Tools

You can invoke the Inspector by calling the control program, or by calling the tools individually. The invocations are similar. As an example we use the control program.

Invocation syntax on the command line

insp_cctc [[*option*]... [*file*]...]...

By default all existing detectors are enabled within the Inspector and there are options that allow you to enable and disable particular detectors.

You can find a detailed description of all Inspector specific tool options in Chapter 4, *Tool Options*.

Detect/ignore issue detectors

With the options **--detect** and **--ignore** you can make a selection of the issue detectors. These options are the same for all Inspector tools.

Detect by means of assembly compare

For some issue detectors where the normal detection gives a potential occurrence of an issue, you can run an extra detector with option **--detect-asm**. This option is only available for the C compiler and by means of the control program and can only detect one issue at a time. With this option the Inspector tool clarifies detection of a certain issue by comparing the assembly source files affected and not affected by that issue after its potential occurrence has been detected.

Detection messages

The following messages can be generated by an Inspector tool:

```
[INSP] detected occurrence of issue id
```

for issues that definitely impact the tool output (no false positives possible).

```
[INSP] detected potential occurrence of issue id
```

for issues that potentially impact the tool output (false positives are possible).

When an assembly difference is detected the following messages can be generated:

```
[INSP] detected change in assembly listing for command: argv
[INSP] asm cmp: assembly listing copies created for analysis.
       original: asm1 fixed: asm2
```

where, *argv* is the tool invocation line and *asm1* and *asm2* are the names of the two differing assembly outputs.

Example

To detect if the C compiler issues `TCVX-43102` and `TCVX-43543` are present and ignore the other detectors, type:

```
insp_cctc --detect=TCVX-43102,TCVX-43543 test.c
```

3.3. Detection Processing of More Complex Issues

The following sections give guidance on how to process more complex and not always obvious issues that the Inspector can detect.

3.3.1. Issue Detector for TCVX-43102

Issue Description

Optimization of struct return may lead to overlapping struct copy.

Detection Processing

This issue is detected when the compiler optimizes return of a struct from a function and ignores potential overlap between the source and the destination. Such overlap may only occur with union containing struct members.

Verify that there is no such overlap possible at the indicated source position. Note that inlining of such a function may lead to it being optimized into a direct assignment.

3.3.2. Issue Detector for TCVX-43543

Issue Description

Sizeof operator applied to a VLA involving variable post-modification causes wrong code.

Detection Processing

This issue is reported when the compiler ignores post-increment/post-decrement operations in the context of sizeof.

Verify that the indicated value's modification does not affect program behavior (e.g. the value is not used afterwards).

You can use control program option **--detect-asm=TCVX-43543** for further assistance. The difference in the generated assembly code indicates potentially problematic instructions. Absence of such a difference means that detection is a false positive.

3.3.3. Issue Detector for TCVX-43587

Issue Description

GLO tracker optimization problem for uninitialized variable

Detection Processing

This issue is reported when the compiler removes the load operation of an uninitialized variable.

You can use control program option **--detect-asm=TCVX-43587** to identify the load instruction removed by the compiler. The difference in the generated assembly code should be checked to verify that the removed load instruction was applied to an uninitialized variable. In this case, the generated code is affected by the issue.

3.3.4. Issue Detector for TCVX-43704

Issue Description

Non justified if condition optimization.

Detection Processing

This issue is reported when the compiler removes the single-statement body of an if-statement (indicated by the source position) with a possible side-effect in condition.

Verify that evaluation of the condition does not invalidate removal of the if-statement body.

3.3.5. Issue Detector for TCVX-43893

Issue Description

C compiler omits value assignment to pointer type function argument with forward store optimization enabled.

Detection Processing

This issue is reported when the compiler misses potential memory aliasing/overlapping and assumes that memory access (store or function call) indicated by the source position is safe to optimize.

Verify that the indicated memory accesses do not overlap with each other and are not incorrectly optimized away.

You can use control program option **--detect-asm=TCVX-43893** for further assistance. The difference in the generated assembly code indicates potentially problematic instructions. Absence of such a difference means that detection is a false positive.

Note that mitigations listed for this issue may still produce false positive detections.

3.3.6. Issue Detector for TCVX-43928

Issue Description

Incorrect reordering of volatile memory reads.

Detection Processing

This issue is reported when the compiler incorrectly reorders volatile memory accesses, indicated by the source positions.

Verify that these memory accesses are not interdependent and that the new order does not change program behavior.

3.3.7. Issue Detector for TCVX-43998

Issue Description

Invalid constant propagation with triple indirection.

Detection Processing

This issue is detected when the compiler overlooks potential memory overlap/aliasing for multiple-indirection memory access indicated by a source position. This may result in various optimizations being applied incorrectly and changing program behavior in unsafe way.

Verify that the indicated indirect memory access is safe and any optimizations applied to it do not change program behavior.

You can use control program option **--detect-asm=TCVX-43998** for further assistance. The difference in the generated assembly code indicates potentially problematic instructions. Absence of such a difference means that detection is a false positive.

3.3.8. Issue Detector for TCVX-44102

Issue Description

Loop invariant code optimization issue.

Detection Processing

This issue is reported during loop-invariant code motion, when an indirect memory access (indicated by the source position) is moved out of the loop and there is a possibility that the corresponding pointer may be uninitialized or incorrectly initialized at its new location.

Verify that the indicated indirect memory access may be correctly and unconditionally dereferenced outside of the containing loop.

3.3.9. Issue Detector for TCVX-44237

Issue Description

Illegal double word access to SFR register range.

Detection Processing

This issue is detected when the compiler removes a volatile modifier from the variable or ignores this modifier during a peephole optimization. A lost volatile modifier may also affect subsequent optimizations.

Verify that accesses to the indicated volatile variable are not incorrectly optimized and remain in order with other volatile accesses.

You can use control program option **--detect-asm=TCVX-444237** for further assistance. The difference in the generated assembly code indicates potentially unsafe optimizations/reorderings. Absence of such a difference means that detection is a false positive.

3.3.10. Issue Detector for TCVX-44278

Issue Description

C++ compiler: generated code results in address 0x0000000 access causing bus trap.

Detection Processing

To verify if a reported potential issue is an actual problem, find the symbol name in the generated intermediate C code (`.ic` file) or in the original C++ code, then check the value in the ELF file using e.g. **elfdump** or in the linker map file.

Note that the linker and **hldumptc** have an option (**-P**/**--print-mangled-symbols**) to print non-demangled symbol names, which match the names in the generated C code. Note also that the line number printed in the issue detection message needs to be matched to a `#line` entry for the correct source file in the generated C code, but the specific line may not have such an entry - the first lower C++ source line number with a `#line` entry needs to be used in that case. Only if the symbol value is zero, then an instance of the problem has been detected.

3.3.11. Issue Detector for TCVX-44387

Issue Description

Erroneous code in code compaction function leads to invalid function parameter.

Detection Processing

This issue is reported when the compiler reorders instructions in a potentially unsafe way. It can only affect functions named `.cocofun_*` generated by the code compaction optimization. The source position in the message indicates the first instruction of the corresponding function. These functions are usually small in size.

Verify that the order of the instructions in the indicated function does not break data dependency.

You can use control program option **--detect-asm=TCVX-44387** for further assistance. The difference in the generated assembly code indicates which instructions have been reordered and should be looked at.

3.3.12. Issue Detector for TCVX-44400

Issue Description

Wrong value is loaded into a 48-bit struct if used as a member of a larger 64-bit struct.

Detection Processing

The Inspector issues a warning every time an incorrect pattern is used. Whether it leads to an actual bug depends on previous initializations and sometimes run-time. An incorrect pattern is only used when a 40-bit or 48-bit structure is assigned as a member of a 64-bit structure with a non-zero offset and this operation is performed on extended registers. The generated code can produce invalid results if other members of the 64-bit (destination) structure have been initialized with non-zero value prior to the assignment in question. If at the moment of the assignment in question other members of the 64-bit structure are uninitialized or initialized with zero, the final result will be correct. Thus one possible mitigation would be to change the order of assignments.

3.3.13. Issue Detector for TCVX-44407

Issue Description

C compiler front-end may produce imprecise FP result (±1 bit difference).

Detection Processing

This issue is detected during the constant folding when the result of the division of a floating-point constant by another constant deviates from the expected value in digits beyond the corresponding type precision. Whether this will affect the resulting constant in the assembly code depends on the whole constant expression being folded. This deviation may affect rounding, leading to the incorrect result of the division itself. It may also be accumulated or amplified by subsequent operations, affecting the result of the whole expression. On the other hand, subsequent operations (e.g. another division or cast to a smaller precision) may negate this deviation. The source position indicates the operation in question, when it is possible.

Verify that the value of the folded constant expression in the assembly code is correct.

You can use control program option **--detect-asm=TCVX-44407** for further assistance. The difference in the generated assembly code indicates incorrectly folded constants. Absence of such a difference means that detection is a false positive.

The mitigation listed in the issue does not work for unnamed constants, but this can be amended by naming them.

3.3.14. Issue Detector for TCVX-44419

Issue Description

Linker does not insert `alignment_protection` section when `copy_unit` is greater than 1.

Detection Processing

To check that a reported potential issue is an actual problem, find the reported section in the linker map file, or in the ELF file using **elfdump** or **hldumptc**. The required alignment protection section "`.alignment_protection`" will not be present after that section, but it may be located in a memory mirror at the correct location in memory. If the alignment protection section is not found in a memory mirror, then an instance of the problem was detected.

3.3.15. Issue Detector for TCVX-44522

Issue Description

The `__dsync()` intrinsic does not always work as a memory fence.

Detection Processing

This issue is reported every time when the `dsync` instruction is used by the C compiler and not properly treated as a memory fence. This may affect code generation, but not necessarily result in incorrect code.

You can use control program option **--detect-asm=TCVX-44522** for further assistance. The difference in the generated assembly code should be checked for memory operations that were optimized away or moved over `dsync` as a result of the missing memory fence. Absence of such a difference means that detection is a false positive.

3.3.16. Issue Detector for TCVX-44737

Issue Description

Compiler generates wrong code.

Detection Processing

This issue is reported when the compiler removes certain conversion operations that might affect the converted value.

You can use control program option **--detect-asm=TCVX-44737** to identify the conversion removed by the compiler. The difference in the generated assembly code should be checked to verify that this conversion does not affect the converted value.

3.3.17. Issue Detector for TCVX-44796

Issue Description

FPU instructions may corrupt 64-bit integer computations.

Detection Processing

This issue is reported when ALU flags are corrupted by FPU instructions.

In addition to the detection message, the Inspector marks detected errors in the generated assembly file with comments to the instructions.

Example:

```
addx d2,d4,d6
ld.w d0,[a10]
ld.w d15,[a10]4
add.f d15,d0,d15 ; Issue TCVX-44796 detected: Carry flag corrupted
```

```
st.w flt,d15
addc d3,d5,d7 ; Carry flag is used here
st.d ll,e2
ret
```

When the code compaction optimization is enabled (option **-Or**), liveness analysis may show false usage (reading) of ALU flags within CoCo functions, leading to a false positive detection.

Example:

```
itof d8,d8 ; Issue TCVX-44796 detected: Overflow flag corrupted
movh.a a14,#@his(.L9)
lea a14,[a14]@los(.L9)
j .cocofun_3 ; Overflow flag is used here
```

In the above example, the overflow flag is not actually used in the CoCo function and this is a false positive. If the Inspector added a "used here" comment to a CoCo function, an additional step is required for verification: build the same module without code compaction (**-OR**). If there is no detection, then it was a false positive.

The Inspector can only detect the formal usage of a flag. After detection, verify that the corrupted flag is meaningfully used by the indicated instruction. In general, the carry flag is almost always used for real and the overflow flag is only used non-formally in rare occasions related to fixed-point calculations.

3.4. Detecting Issues that Cannot be Detected at Compile Time

The following section gives guidance on how to detect issues that cannot be detected at compile time.

3.4.1. Guidance for Detecting Issue TCVX-44325

Issue Description

User stack pointer 4-byte aligned when interrupt occurs between FCALL and FRET.

Issue Prevention

This issue only happens when an interrupt is triggered inside a generated "code compaction" function that uses FCALL/FRET, and the interrupt handler calls another function and hardware is set to use common stack for interrupts and check for stack pointer alignment.

This issue has been deemed impossible to detect in a meaningful way at compile time.

To verify that the problem cannot manifest itself, check that at least one of the following is true:

1. Interrupt Stack in hardware is enabled.
2. Stack pointer check in hardware is disabled.
3. There are no FCALL instructions, generated by code compaction or added with inline assembly.
4. There are no function calls (CALL instructions) inside interrupt handlers.

5. If FCALL instructions are only used inside interrupt handlers, only FCALL inside interrupt handlers that allowed to be interrupted are important. FCALL in an interrupt handler without `__enable_/__bisr_()` qualifier and before `__enable()/__bisr()` intrinsic will not enable the problem. And in such case only CALL instructions in handlers with a higher priority are important as only they can be executed between FCALL and FRET.

If none of these is true, then the problem may potentially happen and cause a hardware exception during program execution.

3.4.2. Guidance for Detecting Issue TCVX-44962 and TCVX-44835

Issue Description

Struct alignments for bit-fields not always EABI compliant

Issue Prevention

Due to these issues, when EABI compliance is requested, the C/C++ compiler may use 4-byte alignment for a struct with a bit-field instead of the EABI-mandated 2-byte alignment.

To verify that the problem cannot manifest itself when using bit-fields, either compile all modules with the same C/C++ compiler or verify that the mentioned 4-byte alignment does not cause any problems in the application.

3.4.3. Guidance for Detecting Issue TCVX-45075

Issue Description

Non-trapping 64-bit floating-point emulation does not handle exceptional results correctly

Issue Detailed Description

If the non-trapping floating-point model is used, the compiler does not correctly process 64-bit floating-point basic operations (+, -, /, and *). Incorrect behavior occurs when the result, or intermediate result of the operation is an exceptional value.

Non-trapping 64-bit floating-point arithmetic is applied if:

- compiler option **--fp-model=-trap** is specified and
- compiler option **--fp-model=+float** is not specified and
- data type `double` is used in the application software.

Malfunctioning occurs:

- When the result of an operation should yield a NaN. For example, calculations like "`a = b / 0.0`" where "b" is any normal floating-point number should yield a NaN value, but results in some arbitrary normal floating-point value.

- When the result of an operation should yield INFINITY. For example, "`a = DBL_MAX + DBL_MAX`" results in some arbitrary value.
- When the result of an operation yields a subnormal number which should be subsequently rounded to 0.0 or to DBL_MIN. For example, `nextafter(DBL_MIN, DBL_MAX) - DBL_MIN` results in some arbitrary value.

Issue Prevention

First verify if non-trapping 64-bit floating-point arithmetic is applied to your application. If so, carefully check if the malfunction could occur and apply the necessary countermeasures to prevent them.

Chapter 4. Tool Options

This chapter provides a detailed description of the Inspector specific options for the control program, C++ compiler, C compiler, assembler and linker.

4.1. Control Program Options

The control program **insp_cctc** facilitates the invocation of the various components of the Inspector from a single command line.

This section lists all control program options that are specific to the Inspector. All other options are the same as the control program of the TriCore VX-toolset for TriCore v6.2r2.

The control program processes command line options either by itself, or, when the option is unknown to the control program, it looks whether it can pass the option to one of the other tools. However, for directly passing an option to the C++ compiler, C compiler, assembler or linker, it is recommended to use the control program options **--pass-c++**, **--pass-c**, **--pass-assembler**, **--pass-linker**.

See the other sections for details on the options of the tools.

When you do not specify an option, a default value may become active.

Control program option: --detect

Command line syntax

--detect=*issue*,...

Description

This option allows you to enable only a specific set of issue detectors for Inspector. Detectors not listed in this option will be disabled. This option is mutually exclusive with control program options **--detect-asm** and **--ignore**.

For the list of issues you can detect, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Example

To detect if the C compiler issue `TCVX-43102` is present, type:

```
insp_cctc --detect=TCVX-43102 test.c
```

Related information

Control program option **--detect-asm** (Enable assembler comparison issue detector)

Control program option **--ignore** (Disable issue detectors)

Control program option: --detect-asm

Command line syntax

`--detect-asm=`*`issue`*

Description

With this option the Inspector tool clarifies detection of a certain issue by comparing the assembly source files affected and not affected by that issue after its potential occurrence has been detected. It can only detect one issue at a time. All other detectors will be disabled. This option is mutually exclusive with control program options **--detect** and **--ignore**.

For the list of issues you can detect, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Example

To detect if the C compiler issue `TCVX-43543` is present by comparing the assembly output, type:

```
insp_cctc --detect-asm=TCVX-43543 test.c
```

Related information

Control program option **--detect** (Enable issue detectors)

Control program option **--ignore** (Disable issue detectors)

Control program option: --ignore

Command line syntax

--ignore=*issue*,...

Description

This option allows you to disable a specific set of issue detectors for Inspector. Detectors not listed in this option will be enabled. This option is mutually exclusive with control program options **--detect** and **--detect-asm**.

For the list of issues you can specify, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Example

To ignore the issue detector for C compiler issue `TCVX-43102` and enable all other issue detectors, type:

```
insp_cctc --ignore=TCVX-43102 test.c
```

Related information

Control program option **--detect** (Enable issue detectors)

Control program option **--detect-asm** (Enable assembler comparison issue detector)

Control program option: --insp-log

Command line syntax

`--insp-log=`*`file`*

Description

With this option the control program will add options to the tools to redirect Inspector messages to the specified file. This file is written in append mode, clearing it is the user's responsibility.

Each detection warning is prefixed with the tool's invocation line. The control program passes this option to the C++ compiler, C compiler, assembler and linker.

Example

```
insp_cctc --detect=TCVX-43102 --insp-log=TCVX-43102.log test.c
```

The log file `TCVX-43102.log` will contain the Inspector messages.

Related information

-

4.2. C++ Compiler Options

This section lists all C++ compiler options that are specific to the Inspector. All other options are the same as the C++ compiler of the TriCore VX-toolset for TriCore v6.2r2.

When you do not specify an option, a default value may become active.

The priority of the options is left-to-right: when two options conflict, the first (most left) one takes effect. The **-D** and **-U** options are not considered conflicting options, so they are processed left-to-right for each source file. You can overrule the default output file name with the **--output-file** option.

C++ compiler option: --detect

Command line syntax

--detect=*issue*,...

Description

This option allows you to enable only a specific set of issue detectors for Inspector. Detectors not listed in this option will be disabled. This option is mutually exclusive with C++ compiler option **--ignore**.

For the list of issues you can detect, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Example

To detect if the C++ compiler issue TCVX-44461 is present, type:

```
insp_cptc --detect=TCVX-44461 test.cc
```

Related information

C++ compiler option **--ignore** (Disable issue detectors)

C++ compiler option: --ignore

Command line syntax

--ignore=*issue*,...

Description

This option allows you to disable a specific set of issue detectors for Inspector. Detectors not listed in this option will be enabled. This option is mutually exclusive with C++ compiler option **--detect**.

For the list of issues you can specify, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Example

To ignore the issue detector for C++ compiler issue `TCVX-44461` and enable all other issue detectors, type:

```
insp_cptc --ignore=TCVX-44461 test.cc
```

Related information

C++ compiler option **--detect** (Enable issue detectors)

C++ compiler option: --insp-log

Command line syntax

`--insp-log=`*`file`*

Description

With this option the C++ compiler duplicates Inspector detection messages to the specified file. This file is written in append mode, clearing it is the user's responsibility.

Each detection warning is prefixed with the tool's invocation line.

Example

```
insp_cptc --detect=TCVX-44461 --insp-log=TCVX-44461.log test.cc
```

The log file `TCVX-44461.log` will contain the Inspector messages.

Related information

-

4.3. C Compiler Options

This section lists all C compiler options that are specific to the Inspector. All other options are the same as the C compiler of the TriCore VX-toolset for TriCore v6.2r2.

When you do not specify an option, a default value may become active.

C compiler option: --detect

Command line syntax

--detect=*issue*,...

Description

This option allows you to enable only a specific set of issue detectors for Inspector. Detectors not listed in this option will be disabled. This option is mutually exclusive with C compiler options **--detect-asm** and **--ignore**.

For the list of issues you can detect, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Example

To detect if the C compiler issue `TCVX-43102` is present, type:

```
insp_ctc --detect=TCVX-43102 test.c
```

Related information

C compiler option **--detect-asm** (Enable assembler comparison issue detector)

C compiler option **--ignore** (Disable issue detectors)

C compiler option: --detect-asm

Command line syntax

`--detect-asm=`*`issue`*

Description

With this option the Inspector tool clarifies detection of a certain issue by comparing the assembly source files affected and not affected by that issue after its potential occurrence has been detected. It can only detect one issue at a time. All other detectors will be disabled. This option is mutually exclusive with C compiler options **--detect** and **--ignore**.

For the list of issues you can detect, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

When an assembly difference is detected the following messages can be generated:

```
W994: [INSP] detected change in assembly listing for command: argv
I992: [INSP] asm cmp: assembly listing copies created for analysis.
             original: asm1 fixed: asm2
```

where, *argv* is the tool invocation line and *asm1* and *asm2* are the names of the two differing assembly outputs.

Example

To detect if C compiler issue `TCVX-43543` is present by comparing the assembly output, type:

```
insp_ctc --detect-asm=TCVX-43543 test.c
```

Related information

C compiler option **--detect** (Enable issue detectors)

C compiler option **--ignore** (Disable issue detectors)

C compiler option: --ignore

Command line syntax

--ignore=*issue*,...

Description

This option allows you to disable a specific set of issue detectors for Inspector. Detectors not listed in this option will be enabled. This option is mutually exclusive with C compiler options **--detect** and **--detect-asm**.

For the list of issues you can specify, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Example

To ignore the issue detector for C compiler issue `TCVX-43102` and enable all other issue detectors, type:

```
insp_ctc --ignore=TCVX-43102 test.c
```

Related information

C compiler option **--detect** (Enable issue detectors)

C compiler option **--detect-asm** (Enable assembler comparison issue detector)

C compiler option: --insp-log

Command line syntax

`--insp-log=`*`file`*

Description

With this option the C compiler duplicates Inspector detection messages to the specified file. This file is written in append mode, clearing it is the user's responsibility.

Each detection warning is prefixed with the tool's invocation line.

Example

```
insp_ctc --detect=TCVX-43102 --insp-log=TCVX-43102.log test.c
```

The log file `TCVX-43102.log` will contain the Inspector messages.

Related information

-

4.4. Assembler Options

This section lists all assembler options that are specific to the Inspector. All other options are the same as the assembler of the TriCore VX-toolset for TriCore v6.2r2.

When you do not specify an option, a default value may become active.

Assembler option: --detect

Command line syntax

--detect=*issue*,...

Description

This option allows you to enable only a specific set of issue detectors for Inspector. Detectors not listed in this option will be disabled. This option is mutually exclusive with assembler option **--ignore**.

For the list of issues you can detect, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Related information

Assembler option **--ignore** (Disable issue detectors)

Assembler option: --ignore

Command line syntax

--ignore=*issue*,...

Description

This option allows you to disable a specific set of issue detectors for Inspector. Detectors not listed in this option will be enabled. This option is mutually exclusive with assembler option **--detect**.

For the list of issues you can specify, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Related information

Assembler option **--detect** (Enable issue detectors)

Assembler option: --insp-log

Command line syntax

`--insp-log=`*`file`*

Description

the assembler duplicates Inspector detection messages to the specified file. This file is written in append mode, clearing it is the user's responsibility.

Each detection warning is prefixed with the tool's invocation line.

Example

`insp_astc --detect=TCVX-`*`xxxx`* `--insp-log=insp_astc.log test.src`

The log file `insp_astc.log` will contain the Inspector messages.

Related information

-

4.5. Linker Options

This section lists all linker options that are specific to the Inspector. All other options are the same as the linker of the TriCore VX-toolset for TriCore v6.2r2.

When you do not specify an option, a default value may become active.

Linker option: --detect

Command line syntax

--detect=*issue*,...

Description

This option allows you to enable only a specific set of issue detectors for Inspector. Detectors not listed in this option will be disabled. This option is mutually exclusive with linker option **--ignore**.

For the list of issues you can detect, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Example

To detect if the linker issue `TCVX-40469` is present, type:

```
insp_ltc --detect=TCVX-40469 test.o
```

Related information

Linker option **--ignore** (Disable issue detectors)

Linker option: --ignore

Command line syntax

--ignore=*issue*,...

Description

This option allows you to disable a specific set of issue detectors for Inspector. Detectors not listed in this option will be enabled. This option is mutually exclusive with linker option **--detect**.

For the list of issues you can specify, see the release notes of the TASKING TriCore v6.2r2 Inspector. When you specify a wrong issue, the list of issues is also listed on the command line.

Example

To ignore the issue detector for linker issue `TCVX-40469` and enable all other issue detectors, type:

```
insp_ltc --ignore=TCVX-40469 test.o
```

Related information

Linker option **--detect** (Enable issue detectors)

Linker option: --insp-log

Command line syntax

`--insp-log=`*`file`*

Description

With this option the linker duplicates Inspector detection messages to the specified file. This file is written in append mode, clearing it is the user's responsibility.

Each detection warning is prefixed with the tool's invocation line.

Example

```
insp_ltc --detect=TCVX-40469 --insp-log=TCVX-40469.log test.o
```

The log file `TCVX-40469.log` will contain the Inspector messages.

Related information

-