EB tresos® IOC safety manual

Date: 2018-10-02, Document version V0.12, Status: RELEASED

EB tresos® IOC safety manual

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

Technical support

Europe	Japan	USA
Phone: +49 9131 7701 6060	Phone: +81 3 5577 6110	Phone: +1 888 346 3813

Support URL

https://www.elektrobit.com/support

Legal notice

Confidential and proprietary information

Table of Contents

1. Document history

The revisions of the document as a whole are documented here. Each revision contains many smaller changes to the individual parts that go into the document, often by different contributors. The exact change history for each part is maintained in a revision control system.

Version	Date	State	Description
V0.1	2016-11-16	PROPOSED	Initial version.
V0.2	2016-12-01	PROPOSED	Incorporate rework resulting from inspection.
V0.3	2016-12-02	PROPOSED	Incorporate rework resulting from TE review.
V0.4	2016-12-06	PROPOSED	Incorporate further rework resulting from rework check.
V0.4	2016-12-06	RELEASED	The document has passed its verification; see [IOCSMINSP1].
V0.5	2017-03-01	PROPOSED	▶ Extend section *Robustness against input errors*. ▶ Add warning about possible deadlocks to section *Configuration of mutual exclusion*. ▶ Describe spinlock type more precisely in VC.IOC_CHCFG_H.Struct.WriterLockXcore.Thread.Lock. ▶ Add VC.IOC_StableArguments.Sender.
V0.6	2017-03-03	PROPOSED	Rework after document inspection.
V0.6	2017-03-03	RELEASED	The document has passed its verification; see [IOCSMINSP2].
V0.7	2017-05-04	PROPOSED	Turned VC.IOC_CHCFG_H.Struct.WriterLockXcore.Thread.Lock into a TRICORE-specific verification criterion and renamed it to VC.TRICORE.IOC_CHCFG_H.Struct.WriterLockXcore.Thread.Lock.
V0.7	2017-06-22	RELEASED	The document has passed its verification; see [IOCSMINSP3].
V0.8	2018-08-10	PROPOSED	The verification criterion VC.IOC_CHCFG_H.IOC_CFG_NREADABLEMKREGIONS was added.
V0.8	2018-08-31	RELEASED	The document has passed its verification; see [IOCSMINSP4].
V0.9	2018-09-17	PROPOSED	The verification criterion VC.IOC_Memory.barrier was added.
V0.10	2018-09-28	PROPOSED	Rework after document inspection.
V0.10	2018-09-28	RELEASED	The document has passed its verification; see [IOCSMINSP5].
V0.11	2018-10-02	PROPOSED	Rework after TE review.
V0.12	2018-10-02	PROPOSED	Further rework.
V0.12	2018-10-02	RELEASED	The document has passed its verification; see [IOCSMINSP5].

Table 1.1. Document history

2. Document information

The *inter-OS-application communicator* (IOC) is specified by AUTOSAR as part of the operating system. Due to its complexity and its dependence on generated code, the IOC implementation for EB tresos Safety OS is provided as an add-on module called EB tresos IOC.

This document is the safety manual for EB tresos IOC.

2.1. Objective

The objective of this document is to provide you with all the information necessary to ensure that EB tresos IOC is used in a safe way in your project.

2.2. Scope and audience

This manual describes the use of EB tresos IOC in system applications which have safety allocations up to ASIL-D. It is valid for all projects and organizations which use EB tresos IOC in a safety-related environment.

The intended audience of this document consists of:

- Software architects
- Safety engineers
- Application developers
- Software integrators

The persons with these roles are responsible for performing the verification methods defined in this manual. They shall have the following knowledge and abilities:

- C-programming skills, especially in embedded systems.
- Experience in programming AUTOSAR-compliant ECUs.

Experience with safety standards and software development in safety-related environments is recommended.

2.3. Motivation

This manual provides information on how to use EB tresos IOC correctly in safety-related projects. If EB tresos IOC is used differently, the resulting system might not comply with the assumed requirements of EB tresos IOC that are defined in Chapter 5, “Application constraints and requirements“. You must take appropriate actions to ensure that your safety requirements are fulfilled.

2.4. Structure

Chapter 2, “Document information“ (this chapter) gives a brief description of the document structure.

Chapter 3, “About EB tresos IOC“ describes EB tresos IOC and its features.

Chapter 4, “Using EB tresos IOC safely“ describes how to use EB tresos IOC safely.

Chapter 5, “Application constraints and requirements“ describes the application constraints and the assumed requirements.

Chapter 6, “Configuration verification criteria“ lists the criteria that you must use to verify that EB tresos IOC is configured correctly for safe use. Safe use can only be determined for the *item* in which EB tresos IOC is used. You must perform a hazard analysis and risk assessment for the item. See [ISO26262-3_1ST], clause 7.

Appendix A, “Reserved symbols“ lists the program identifiers that are reserved for use by EB tresos IOC.

Appendix B, “Safety of services“ lists the API services provided by EB tresos IOC and the freedom from interference provided for each service.

Finally, the Bibliography lists the documents that are referenced in the text.

2.5. Typography and style conventions

Throughout the documentation you see that words and phrases are displayed in bold or italic font, or in Monospace font. To find out what these conventions mean, consult the following table. All default text is written in Arial Regular font without any markup.

Convention	Item is used	Example
Arial italics	to define new terms	The *basic building blocks* of a configuration are module configurations.
Arial italics	to emphasize	If your project's release version is mixed, all content types are available. It is thus called *mixed version*.
Arial italics	to indicate that a term is explained in the glossary	...exchanges *protocol data unit*s (*PDU*s) with its peer instance of other ECUs.
Arial boldface	for menus and submenus	Choose the **Options** menu.
Arial boldface	for buttons	Select **OK**.

Convention	Item is used	Example
Arial boldface	for keyboard keys	Press the **Enter** key
Arial boldface	for keyboard combination of keys	Press **Ctrl+Alt+Delete**
Arial boldface	for commands	Convert the XDM file to the newer version by using the **legacy convert** command.
Monospace font (Courier)	for file and folder names, also for chapter names	Put your script in the `function_name/abc-folder`
Monospace font (Courier)	for code	`for (i=0; i<5; i++) { /* now use i */ }`
Monospace font (Courier)	for function names, methods, or routines	The `cos` function finds the cosine of each array element. Syntax line example is `MLGetVar ML_var_name`
Monospace font (Courier)	for user input/indicates variable text	Enter a `three-digit prefix` in the menu line.
Square brackets [ ]	for optional parameters; for command syntax with optional parameters	`insertBefore [<opt>]`
Curly brackets {}	for mandatory parameters; for command syntax with mandatory parameters (in curly brackets)	`insertBefore {<file>}`
Three dots …	for further parameters	`insertBefore [<opt>…]`
A vertical bar |	to separate parameters in a list from which one parameters must be chosen or used; for command syntax, indicates a choice of parameters	`allowinvalidmarkup {on|off}`
Warning	to show information vital for the success of your configuration	**WARNING** **This is a warning** This is what a warning looks like.
Notice	to give additional important information on the subject	**NOTE** **This is a notice** This is what a notice looks like.
Tip	to provide helpful hints and tips	**TIP** **This is a tip** This is what a tip looks like.

3. About EB tresos IOC

EB tresos IOC implements a subset of the inter-OS-application communicator that is specified in the AUTOSAR OS specification.

You can use EB tresos IOC in projects where a safety integrity level up to ASIL-D is demanded.

You can use EB tresos IOC to provide communication between software components that need to be free from spatial interference with each other.

EB tresos IOC consists of two parts:

- A set of functions and macros that execute in the context of the thread that calls them.
- A set of functions that execute in the context of the EB tresos Safety OS microkernel.

3.1. Architecture of the surrounding system

The *surrounding system* is defined as the hardware and software that comprises the ECU, excluding EB tresos IOC.

EB tresos IOC does not depend on the architecture of the surrounding hardware but it uses features of the microcontroller on which it runs. The use of these features is encapsulated within EB tresos IOC and does not need hardware-specific treatment in this document.

EB tresos IOC is designed to be used with EB tresos Safety OS. It requires the spatial freedom from interference provided by EB tresos Safety OS as well as the add-on feature of the microkernel. The API provided by EB tresos IOC is designed to be used by EB tresos Safety RTE. You can also use it with other RTE implementations provided that appropriate measures are taken to verify that EB tresos IOC is used correctly.

3.2. Description of EB tresos IOC

3.2.1. Identification of EB tresos IOC

The header files and source files that belong to EB tresos IOC begin with the prefix `Ioc_`. The only exception to this rule is the header file `Ioc.h`.

3.2.2. Robustness of EB tresos IOC

3.2.2.1. Robustness against hardware faults

EB tresos IOC is susceptible to hardware faults. It is assumed that the hardware uses fault detection mechanisms such as dual redundant processing (e.g. *lockstep mode*), *error detection and correction codes* (ECC) etc.

3.2.2.2. Robustness against systematic software errors

EB tresos IOC is susceptible to systematic software errors in its own code. Although EB tresos IOC contains some checks of the validity of its internal variables, it cannot be guaranteed that a systematic error in EB tresos IOC is detected.

To guard against this risk, EB tresos IOC is developed using the stringent practices required for ASIL-D as demanded by [ISO26262_1ST].

3.2.2.3. Robustness against configuration errors

EB tresos IOC is sensitive to configuration errors. EB tresos IOC accepts a wide variety of possible configurations. EB tresos IOC has only very limited support to detect invalid configurations. In particular, many configurations that are incorrect for a given system may be valid from EB tresos IOC's point of view.

You must verify the configuration presented to EB tresos IOC as described in Chapter 6, “Configuration verification criteria“. The configuration must meet the safety requirements of the surrounding system.

3.2.2.4. Robustness against resource conflicts

When so configured, EB tresos IOC uses mutual exclusion locks to protect data against concurrent access. Provided that the locks are correctly configured and (in the case of cross-core locks) IOC's locks are not used outside the IOC API, EB tresos IOC is robust against resource conflicts. To verify that your lock configuration is correct, see Section 6.8.2, “Configuration of mutual exclusion”.

3.2.2.5. Robustness against interrupt overload

EB tresos IOC does not make use of interrupts, so it is not susceptible to interrupt overload.

3.2.2.6. Robustness against input errors

EB tresos IOC is merely a conduit for the data that it transfers. It is not affected by errors in that data.

The integrity of the data in a particular channel can be compromised by the applications that are permitted to write to that channel. In particular the following problems may arise:

- The transferred data may be incorrect if the sender behaves unexpectedly.
- The transferred data of one sender may be incorrect, if multiple senders are present and one sender behaves unexpectedly.
- The sender may be blocked within the communication if multiple senders are present and one sender behaves unexpectedly.
- The receiver may be blocked if one sender behaves unexpectedly.

Robustness against input errors is outside the scope of EB tresos IOC.

3.2.2.7. Robustness against data starvation

EB tresos IOC is merely a conduit for the data that it transfers. It is not affected by starvation of that data. The surrounding application is responsible for detecting stasis in unqueued channels and lack of input in queued channels.

3.2.3. What EB tresos IOC does not do

EB tresos IOC does not in itself provide safety. It is a mechanism that you can use to allow your safety-related software components to communicate with each other without interference from other software components.

EB tresos IOC does not provide end-to-end verification measures, such as error-detecting codes and alive supervision, that you need for communication over an unsafe channel. EB tresos IOC assumes that its channels are protected by the operating system.

3.3. Outstanding anomalies

The EB tresos® IOC User's Guide [IOCUSRDOC] contains deviations from the AUTOSAR standard. You can find information about all known issues for a particular version of EB tresos IOC in EB tresos AutoCore known problems (EB_tresos_Autocore_known_issues-SAFETYOS_<version>_COMMON_SRC.pdf) [KNOWNPROBLEMS] corresponding to your release. It is updated regularly.

3.4. Backward compatibility

This is the first version of EB tresos IOC. There is no backward compatibility.

3.5. Compatibility with other systems

EB tresos IOC is designed and tested for use with EB tresos Safety OS. Use only the versions that are part of your release together. No other combinations are supported.

The EB tresos AutoCore Quality Statement Safety OS [Q-STATEMENT] documents the supported toolchain and its configuration.

3.6. Change control

Contact the EB Product Support and Customer Care Team to initiate a change request. You can find contact details at the front of this document.

4. Using EB tresos IOC safely

Before you use EB tresos IOC you must do the following:

- Read and understand the EB tresos Safety OS safety manual [SAFETYMAN].
- Fulfill all requirements that are stated in the EB tresos Safety OS safety manual.

The EB tresos IOC user's guide describes how to install EB tresos IOC. The EB tresos Safety OS safety manual describes how to implement and build your system and how to verify that the binary image of your system contains what you expect.

4.1. Prerequisites

For safe operation, EB tresos IOC relies on the freedom from interference provided by EB tresos Safety OS. Use with any other operating system is not supported by this safety manual.

EB tresos IOC is designed for use with EB tresos Safety RTE. If you use EB tresos IOC with a different RTE you must verify that the IOC API is called correctly.

You must determine the freedom from interference requirements for your runnables and allocate the runnables to separate tasks or ISRs in order to fulfill your freedom from interference requirements.

4.2. Correct use and integration of EB tresos IOC with EB tresos Safety OS

You must ensure that the memory protection features of your hardware and of EB tresos Safety OS are correctly configured according to your functional and safety requirements.

The details of this configuration are outside the scope of this safety manual, but you can find some general guidelines in Section 6.8, “Verification criteria for the operating system”.

4.3. Steps outside the scope of Elektrobit Automotive GmbH

The architecture, design, implementation, verification and validation of your system are outside the scope of this safety manual. It is the system designer's responsibility to ensure that the system as a whole is designed in accordance with the requirements and that the EB tresos IOC configuration correctly implements the design.

5. Application constraints and requirements

5.1. Safety Element out of Context (SEooC) definition

EB tresos IOC is developed as a Safety Element out of Context (SEooC) according to [ISO26262-10_1ST].

5.1.1. Functional scope of the SEooC

5.1.1.1. Provided functionality

The functional scope of the SEooC is defined by the requirements as specified in Section 5.3, “Level 1 requirements” and Section 5.4, “Level 2 requirements”. All requirements that have safety class ASIL-D are within the scope of the safety-related part of EB tresos IOC.

WARNING ⚠

Additional safety mechanisms may be necessary

The requirements listed in these sections may not be sufficient to fulfill all safety requirements imposed on the software architecture of the entire system. Depending on the application, additional safety mechanisms which are not part of EB tresos IOC may be necessary to guarantee system safety.

5.1.1.2. Assumed hardware functionality

EB tresos IOC assumes that certain functionality is provided by the hardware. These assumptions are listed in Section 5.5, “Assumptions on the environment”.

5.1.1.3. Assumed employment

EB tresos IOC assumes that it is employed according to the specification given in the safety manual.

5.1.1.4. Assumed compiler confidence level

EB tresos IOC assumes that the user uses a compiler in a version and with compiler settings that Elektrobit Automotive GmbH supports for a particular release.

EB tresos IOC further assumes that the user takes appropriate measures to create confidence in the compiler (e.g. tool qualification).

5.1.1.5. Assumed external functionality

EB tresos IOC provides functionality through which **PARTIAL** freedom from interference on the communication level w.r.t. memory can be ensured.

EB tresos IOC relies on an operating system that provides spatial freedom from interference.

To ensure freedom from interference on other levels it is assumed that the user takes additional measures within the entire software architecture surrounding EB tresos IOC. Potential measures are the following:

- a software component that provides control flow and deadline monitoring for the supervision of the temporal domain, and/or
- a software component that provides the communication within the ECU and to other ECUs (end-to-end protection)

The type and implementation of these protection mechanisms depend on the architecture of the application that utilizes functionality of EB tresos IOC.

5.1.2. Implementation scope of the SEooC

The safety-related part of EB tresos IOC consists of the following components:

- Source code in the Ioc_TS_<ID> plugin as installed in EB tresos Studio.
- The EB tresos IOC configuration, written according to the safety manual.
- The *quality statement*, which contains the following information on the EB tresos IOC qualification:
  - The compiler and compiler version
  - The compiler options
  - The processor derivative used for testing

The EB tresos IOC generator is not part of the SEooC. The user has to verify the generated configuration manually according to the safety manual.

EB tresos IOC is released as part of either EB tresos AutoCore OS or EB tresos Safety OS. The quality statement is provided for each release, platform, and derivative of the corresponding EB tresos AutoCore OS or EB tresos Safety OS release.

The naming scheme of the quality statement is as follows:

- `EB_tresos_AutoCore-quality_statement-SafetyOS_<Version>-<Derivative>.pdf` for a release of EB tresos Safety OS or
- `EB_tresos_AutoCore-quality_statement-AutoCoreOS-<Version>-<Derivative>.pdf` for a release of EB tresos AutoCore OS.

EB_tresos_AutoCore-quality_statement-SafetyOS_2.0.8-TC29XT.pdf

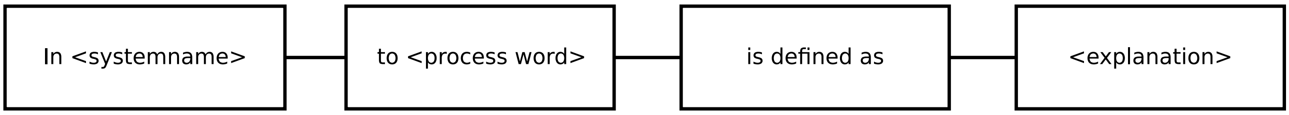
Example 5.1. Example of a quality statement name

5.2. Definitions

This section defines the processes and terms used in Section 5.3, “Level 1 requirements” and Section 5.4, “Level 2 requirements”.

5.2.1. Process words

This section defines all *processes*.



In <systemname>

to <process word>

is defined as

<explanation>

Figure 5.1. Scheme for defining a process

Process	**Definition**
to organize	In EB tresos IOC *to organize* is defined as structuring something according to a defined property.
to provide	In EB tresos IOC *to provide* is defined as making something available.
to support	In EB tresos IOC *to support* is defined as contributing to making something available.

Table 5.1. Definition of processes

5.2.2. Term definitions

This section defines all *terms*.

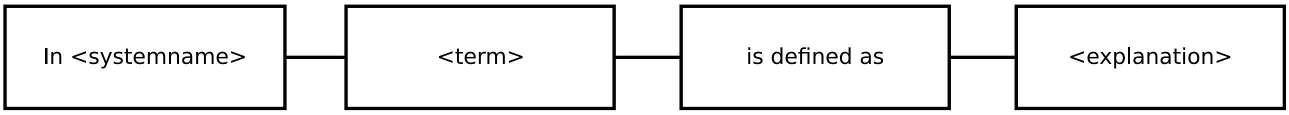


Figure 5.2. Scheme for defining a term

Term	Definition
API	In EB tresos IOC *API* is defined as abstract programming interface that consists of the function name, the parameters and the provided functionality of a C function.
channel	In EB tresos IOC *channel* is defined as a basic mechanism on which communication can proceed.
coherent memory view	In EB tresos IOC *coherent memory view* is defined as the property of a memory address such that a read operation on that address yields the most recently written value irrespective of the processor core from which the operation originates.
communication	In EB tresos IOC *communication* is defined as the exchange of one or more data elements between multiple executable entities.
core-local	In EB tresos IOC *core-local* is defined as the the property of a mechanism to be accessible from multiple executable entities that are executed on the same core of one processor.
cross-core	In EB tresos IOC *cross-core* is defined as the property of a mechanism that affects two or more cores of a multi-core processor.
critical section	In EB tresos IOC *critical section* is defined as a section of an executable entity's internal logic that is executed mutually exclusively with other executable entities. Note: This definition is derived from [ISO24765].
data element	In EB tresos IOC *data element* is defined as data stored in a contiguous segment of the microcontroller's address space.
executable entity	In EB tresos IOC *executable entity* is defined as software executed as an operating system application.
free from interference	In EB tresos IOC *free from interference* is defined as ▶ having *freedom from interference* (when consulting [ISO26262_1ST]) and ▶ having *non-interference between software elements* (when consulting [ISO/IEC 61508:2010]). See also Section 5.2.3, “Term: freedom from interference”.

hardware	In EB tresos IOC *hardware* is defined as the physical equipment used to process, store, or transmit computer programs or data. Note: This definition is copied from [ISO24765].
mutual exclusion	In EB tresos IOC *mutual exclusion* is defined as the problem of ensuring that no competing executable entities can be in their critical section at the same time.
operating system	In EB tresos IOC *operating system* is defined as a collection of software, firmware, and hardware elements that controls the execution of executable entities and provides such services as computer resource allocation and job control in a computer system. Note: This definition is derived from [ISO24765].
platform	In EB tresos IOC *platform* is defined as the combination of hardware, operating system, toolchain and toolchain settings that is used.
queued	In EB tresos IOC *queued* is defined as a property of communication that specifies that data is consumed when it is read. Additionally, the sequential order of reception of data elements is identical to the sequential order of transmission.
reliable execution environment	In EB tresos IOC *reliable execution environment* is defined as a microcontroller that provides a mechanism that prevents or detects non-systematic hardware faults (data corruption, incorrect execution of instructions). Note: Examples are lockstep mode and ECC memory.
unauthorized access	In EB tresos IOC *unauthorized access* is defined as the attempt to access a memory location or hardware component without having the appropriate access rights.
unqueued	In EB tresos IOC *unqueued* is defined as a property of communication that specifies that data is not consumed when it is read. This means that the data elements of the last completed transmission are received.

Table 5.2. Definition of terms

5.2.3. Term: freedom from interference

For your convenience, the following two definitions have been copied verbatim into this document (see the following sections).

5.2.3.1. Freedom from interference in ISO 26262

Definition in [ISO26262-1_1ST]:
absence of cascading failures between two or more elements that could lead to the violation of a safety requirement.

ISO 26262 states (see [ISO26262-6_1ST] Annex D) that freedom from interference has three aspects:

- Timing and execution
- Memory
- Exchange of information

5.2.3.2. Non-interference between software elements in IEC 61508

Definition in [ISO/IEC 61508-3:2010] Annex F.1:
The term "independence of execution" means that elements will not adversely interfere with each other's execution behavior such that a dangerous failure would occur. It is used to distinguish other aspects of independence which may be required between elements, in particular diversity, to meet other requirements of the standard.

Independence of execution should be achieved and demonstrated both in the spatial and temporal domains.

- *Spatial: the data used by one element shall not be changed by another element. In particular, it shall not be changed by a non-safety related element.*
- *Temporal: one element shall not cause another element to function incorrectly by taking too high a share of the available processor execution time, or by blocking execution of the other element by locking a shared resource of some kind.*

5.2.4. Requirements

This section defines the scheme for constructing *requirements*.

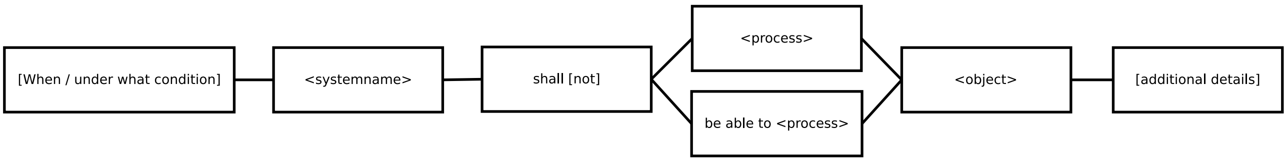


Figure 5.3. Scheme for defining a requirement

5.3. Level 1 requirements

Id:	IOC
Doctype:	`reqspec1`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC shall provide mechanisms for communication.
Safety class:	ASIL-D
Safety rationale:	EB tresos IOC shall be developed to ASIL-D processes because the ECU software (including safety-related elements) is not free from interference from IOC when functions of IOC are used.

Id:	IOC.Channel
Doctype:	`reqspec1`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC shall organize communication in channels.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent specobject.
Provides coverage to:	IOC, Version: 1

5.4. Level 2 requirements

5.4.1. Communication requirements

Id:	IOC.Channel.DataElement
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC shall organize data elements of a communication channel.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Comment:	Every channel configuration consists of one or more data elements.
Provides coverage to:	IOC.Channel, Version: 1

Id:	IOC.Communication.Unqueued
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC shall provide mechanisms for unqueued core-local and unqueued cross-core communication with multiple senders and multiple receivers.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Channel, Version: 1 IOC.Channel.DataElement, Version: 1

Id:	IOC.Communication.Queued
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC shall provide mechanisms for queued core-local and queued cross-core communication with multiple senders and one receiver.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Channel, Version: 1 IOC.Channel.DataElement, Version: 1

Id:	IOC.Channel.PartialUpdate
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	For unqueued communication of a channel with multiple configured data elements, EB tresos IOC shall provide writing of only a subset of the data elements.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Channel, Version: 1 IOC.Channel.DataElement, Version: 1

Id:	IOC.Channel.Element.VariableLength
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	For unqueued and queued communication EB tresos IOC shall provide the ability to transmit a variable length of a data element up to the maximum length of the type of the element.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Channel, Version: 1 IOC.Channel.DataElement, Version: 1

Id:	IOC.API.Unqueued.ReInit
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	For unqueued communication, EB tresos IOC shall provide the following API: ▶ `ioc_return_t IOC_ReInit(ioc_uint32_t channel)`, that resets the configured channel *channel* to a state as if no data elements were transferred.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Communication.Unqueued, Version: 1

Id:	IOC.API.Unqueued.Single
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	For unqueued communication, EB tresos IOC shall provide the following API: ▶ `ioc_return_t IOC_Read(ioc_uint32_t channel, void *data)`, that reads from channel *channel* and stores the data element into *data*. ▶ `ioc_return_t IOC_Write(ioc_uint32_t channel, const void *data)`, that writes the data element *data* to the channel *channel*.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Communication.Unqueued, Version: 1

Id:	IOC.API.Unqueued.Extended
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	For unqueued communication, EB tresos IOC shall provide the following API: ▶ `ioc_return_t IOC_ReadExt(ioc_uint32_t channel, ioc_ilength_t *lengths, void * const *data)`, that reads from channel *channel* and stores the length and data of the elements into the specified locations. ▶ `ioc_return_t IOC_WriteExt(ioc_uint32_t channel, const ioc_extinput_t *data)`, that writes the array of data *data* to the channel *channel*. *data* comprises both the payload and the length of the data.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Communication.Unqueued, Version: 1 IOC.Channel.PartialUpdate, Version: 1 IOC.Channel.Element.VariableLength, Version: 1

Id:	IOC.API.Queued.EmptyQueue
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	For queued communication, EB tresos IOC shall provide the following API: ▶ `ioc_return_t IOC_EmptyQueue(ioc_uint32_t channel)`, that resets the configured channel *channel* to a state as if no data element was transferred.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Communication.Queued, Version: 1

Id:	IOC.API.Queued.Single
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	For queued communication, EB tresos IOC shall provide the following API: ▶ `ioc_return_t IOC_Send(ioc_uint32_t channel, const void *data)`, that writes the data element *data* into the configured channel *channel*. ▶ `ioc_return_t IOC_Receive(ioc_uint32_t channel, void *data)`, that reads from the configured channel *channel* and stores the read data element into *data*.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Communication.Queued, Version: 1

Id:	IOC.API.Queued.Extended
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	For queued communication, EB tresos IOC shall provide the following API: ▶ `ioc_return_t IOC_ReceiveExt(ioc_uint32_t channel, ioc_ilength_t *lengths, void * const *data)`, that reads from channel *channel* and stores the length and data of the elements into the specified locations. ▶ `ioc_return_t IOC_SendExt(ioc_uint32_t channel, const ioc_extinput_t *data)`, that writes the array of data *data* to the channel *channel*. *data* comprises both the payload and the length of the data.
Safety class:	ASIL-D
Safety rationale:	Inherited from parent requirement.
Provides coverage to:	IOC.Communication.Queued, Version: 1 IOC.Channel.Element.VariableLength, Version: 1

Id:	IOC.API.Wrappers
Doctype:	`reqspec2`
Status:	APPROVED
Version:	1
Description:	The IOC shall support AUTOSAR-compliant wrapper functions.
Safety class:	QM
Safety rationale:	The generator for the wrappers resides in EB tresos AutoCore OS which is developed according to QM processes.
Provides coverage to:	IOC.Channel, Version: 1 IOC.Channel.DataElement, Version: 1

5.5. Assumptions on the environment

This section defines all assumptions on the environment.

5.5.1. Defining assumptions

Assumptions must be defined using the following scheme:

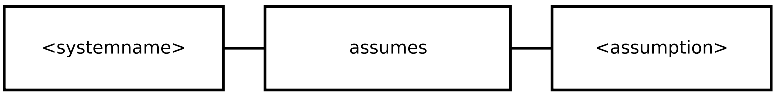


Figure 5.4. Scheme for defining assumptions

5.5.2. Assumptions on the environment

To guarantee the fulfillment of the level 1 and 2 requirements, EB tresos IOC assumes that the environment provides the following features:

NOTE

Assumptions do not have safety integrity levels
Safety class and safety rationale are not specified for assumptions because assumptions do not have safety integrity levels.

Id:	Assumptions.ReliableExecutionEnvironment
Doctype:	`assumptions`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC assumes that the hardware provides a reliable execution environment.

Id:	Assumptions.CoherentMemory
Doctype:	`assumptions`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC assumes that the hardware provides a coherent memory view at all times on a given addresses used by EB tresos IOC.
Comment:	This excludes the use of non-coherent caches for any memory area that EB tresos IOC uses for its communication data.

Id:	Assumptions.MutualExclusion
Doctype:	`assumptions`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC assumes that the hardware provides a mechanism with which mutual exclusion can be established between different cores of the processor.

Id:	Assumptions.DataRepresentation
Doctype:	`assumptions`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC assumes that the platform provides an identical representation of the data elements for all participants of the communication.

Id:	Assumptions.SafetyOS
Doctype:	`assumptions`
Status:	APPROVED
Version:	1
Description:	EB tresos IOC assumes that it is deployed in combination with the EB tresos Safety OS.

5.6. Overview

This section provides an overview over the relationship of requirements on level 1 and 2.

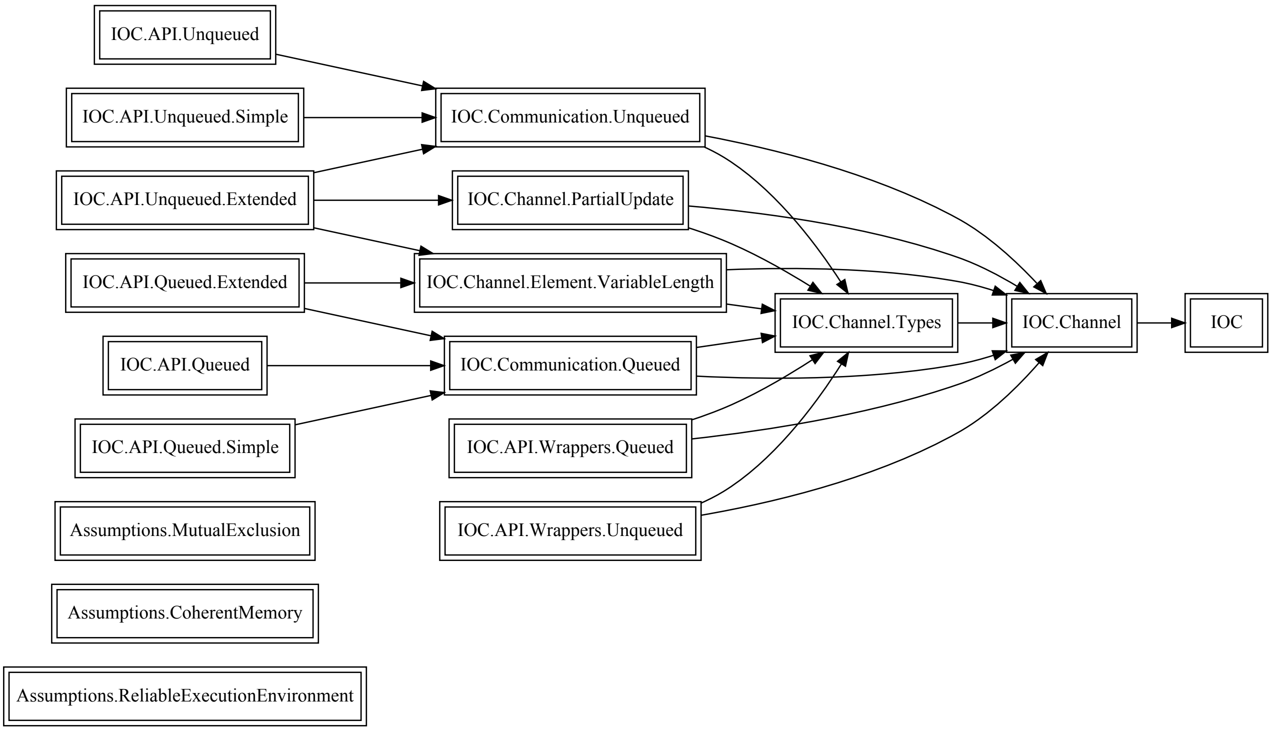

IOC.API.Unqueued
IOC.API.Unqueued.Simple
IOC.API.Unqueued.Extended
IOC.API.Queued.Extended
IOC.API.Queued
IOC.API.Queued.Simple
Assumptions.MutualExclusion
Assumptions.CoherentMemory
Assumptions.ReliableExecutionEnvironment
IOC.Communication.Unqueued
IOC.Channel.PartialUpdate
IOC.Channel.Element.VariableLength
IOC.Communication.Queued
IOC.API.Wrappers.Queued
IOC.API.Wrappers.Unqueued
IOC.Channel.Types
IOC.Channel
IOC

Figure 5.5. Overview of requirements on level 1 and 2

6. Configuration verification criteria

To use EB tresos IOC safely, you need a correct configuration, both of IOC itself and the operating system under which it runs.

EB tresos Studio, and in particular the generator plugin for EB tresos IOC, are not developed to the standards required for safety-relevant applications. For safe operation, you must ensure that the files created by EB tresos Studio specify the intended configuration of the IOC channels that you configured.

This chapter provides the verification criteria for the following generated files:

- `Ioc_gen.h`
- `Ioc_gen_libcfg.h`
- `Ioc_ChCfg.h`
- `Ioc_BufferTypes.h`
- `Ioc_configuration.c`
- `Ioc_gen.c`
- `Ioc_data_app_ch*_Send.c`
- `Ioc_data_kern_c*_ch*_Send.c`
- `Ioc_data_kern_shared_ch*_Send.c`
- `Ioc_data_app_ch*_Receive.c`

You can use these criteria to verify that the generated files accurately represent the IOC configuration provided to the IOC generator. They do not verify that the IOC configuration is correct for any particular application and you cannot use them to do that.

The configuration header files define macros that specify the initial values for constants (typically arrays or primitive scalar values). Each macro is defined at most once.

The IOC API is generated as a set of macros and functions that you call from your application. Each of these APIs is mapped to a call to one of the generic functions provided by IOC. You must verify that the mapping calls the correct generic function with the correct parameters.

The criteria are listed by file. The statement in each criterion says what you must verify. If the generated configuration fails to satisfy even a single criterion, you must not use the configuration in a safety-related system without further confirmation, by means of a separate assessment, that the deviation does not result in an unacceptable risk.

6.1. Verification criteria for Ioc_gen.h

`Ioc_gen.h` contains the channel IDs and the generated API. As well as the read and write methods for each channel, it declares methods for reinitializing unqueued channels and emptying queued channels.

[VC.IOC_GEN_H.ChannelId]

For each configured IOC channel `Xyz`, verify that the channel identifier macro `Ioc_Xyz` is defined to be the index of the channel's configuration in the macro `IOC_CHANNEL_CONFIGURATIONS`.
`IOC_CHANNEL_CONFIGURATIONS` is defined in `Ioc_ChCfg.h`. See also VC.IOC_CHCFG_H.ListOrder.

[VC.IOC_GEN_H.ChannelIdParameter]

Whenever a numerical channel ID is passed to a static API function, verify that the value is the same as the symbolic ID as specified in VC.IOC_GEN_H.ChannelId.

[VC.IOC_GEN_H.ReadMethod]

For each configured IOC channel `Xyz`, verify that there is exactly one of the following macros or function prototypes:

- `IocRead_Xyz()`
- `IocReadGroup_Xyz()`
- `IocReceive_Xyz()`
- `IocReceiveGroup_Xyz()`

The selection depends on how you configured the channel. See the verification criteria VC.IOC_GEN_H.ReadMethod.*
This macro or prototype is referred to as the *read method*.

[VC.IOC_GEN_H.ReadMethod.UnqueuedSingle]

For each configured IOC channel `Xyz` that is configured for unqueued (*last is best*) communication of a single fixed-length data element, verify that the read method is `IocRead_Xyz(arg0)`.
If the method is generated as a function prototype, the parameter shall be a pointer to a variable of the type specified for the channel.
If the method is generated as a macro, the macro shall call `IOC_Read(Ioc_Xyz, arg0)`.

[VC.IOC_GEN_H.ReadMethod.UnqueuedSingleVariable]

For each configured IOC channel `Xyz` that is configured for unqueued (*last is best*) communication of a single variable-length data element, verify that the read method is `IocRead_Xyz(arg0, lengthArg0)`.
If the method is generated as a function prototype, the `arg0` parameter shall be a pointer to a variable of the type specified for the channel, the `lengthArg0` parameter shall be of type `ioc_varlength_t *` and the return type shall be `Std_ReturnType`.

[VC.IOC_GEN_H.ReadMethod.UnqueuedGroup]

For each configured IOC channel `Xyz` that is configured for unqueued ("last is best") communication of a group of data elements, verify that the read method is the function prototype `Std_ReturnType IocReadGroup_Xyz(...)`. The argument parameters shall be pointers to variables of the types specified for the data elements in the channel and shall be in the order specified in the channel. For variable-length data elements the argument parameter shall be immediately followed by a length parameter of type `ioc_varlength_t *`.

[VC.IOC_GEN_H.ReadMethod.QueuedSingle]

For each configured IOC channel `Xyz` that is configured for queued communication of a single fixed-length data element, verify that the read method is `IocReceive_Xyz(arg0)`.
If the method is generated as a function prototype, the parameter shall be a pointer to a variable of the type specified for the channel.
If the method is generated as a macro, the macro shall call `IOC_Receive(Ioc_Xyz, arg0)`.

[VC.IOC_GEN_H.ReadMethod.QueuedSingleVariable]

For each configured IOC channel `Xyz` that is configured for queued communication of a single variable-length data element, verify that the read method is `IocReceive_Xyz(arg0, argLength0)`. If the method is generated as a function prototype, the `arg0` parameter shall be a pointer to a variable of the type specified for the channel, the `lengthArg0` parameter shall be of type `ioc_varlength_t *` and the return type shall be `Std_ReturnType`.

[VC.IOC_GEN_H.ReadMethod.QueuedGroup]

For each configured IOC channel `Xyz` that is configured for queued communication of a group of data elements, verify that the read method is `Std_ReturnType IocReceiveGroup_Xyz(...)`. The argument parameters shall be pointers to variables of the types specified for the data elements in the channel and shall be in the order specified in the channel. For variable-length data elements the argument parameter shall be immediately followed by a length parameter of type `ioc_varlength_t *`.

[VC.IOC_GEN_H.WriteMethod]

For each configured IOC channel `Xyz`, verify that there is exactly one of the following macros or function prototypes:

- `IocWrite_Xyz()`
- `IocWriteGroup_Xyz()`
- `IocSend_Xyz()`
- `IocSendGroup_Xyz()`

The selection depends on how you configured the channel. See the verification criteria VC.IOC_GEN_H.WriteMethod.*
This macro or prototype is referred to as the *write method*.

[VC.IOC_GEN_H.WriteMethod.UnqueuedSingle]

For each configured IOC channel `Xyz` that is configured for unqueued (*last is best*) communication of a single fixed-length data element, verify that the write method is `IocWrite_Xyz(arg0)`. If the method is generated as a function prototype, the parameter shall be of the type specified for the channel and the return type shall be `Std_ReturnType`. For a complex type, the parameter shall be declared `const *`.

[VC.IOC_GEN_H.WriteMethod.UnqueuedSingleVariable]

For each configured IOC channel `Xyz` that is configured for unqueued (*last is best*) communication of a single variable-length data element, verify that the write method is `Std_ReturnType IocWrite_Xyz(arg0, ioc_varlength_t argLength0)`. The `arg0` parameter shall be declared `const *`.

[VC.IOC_GEN_H.WriteMethod.UnqueuedGroup]

For each configured IOC channel `Xyz` that is configured for unqueued (*last is best*) communication of a group of data elements, verify that the write method is `Std_ReturnType IocWriteGroup_Xyz(...)`. The `arg` parameters shall be of the types specified by the data elements in the channel, in the correct order. For primitive data elements the type of the `arg` parameter shall be as specified. For complex data elements the `arg` parameter shall be declared as `const *`. For variable-length data elements the `arg` parameter shall be followed by a length parameter of type `ioc_varlength_t`.

[VC.IOC_GEN_H.WriteMethod.QueuedSingle]

For each configured IOC channel `Xyz` that is configured for queued communication of a single fixed-length data element, verify that the write method is `IocSend_Xyz(arg0)`. If the method is generated as a function prototype, the parameter shall be of the type specified for the channel and the return type shall be `Std_ReturnType`. For a complex type, the parameter shall be declared `const *`.

[VC.IOC_GEN_H.WriteMethod.QueuedSingleVariable]

For each configured IOC channel `Xyz` that is configured for queued communication of a single variable-length data element, verify that the write method is `Std_ReturnType IocSend_Xyz(arg0, ioc_varlength_t argLength0)`. The `arg0` parameter shall be declared `const *`.

[VC.IOC_GEN_H.WriteMethod.QueuedGroup]

For each configured IOC channel `Xyz` that is configured for queued communication of a group of data elements, verify that the write method is `Std_ReturnType IocSendGroup_Xyz(...)`. The `arg` parameters shall be of the types specified by the data elements in the channel, in the correct order. For primitive data elements the type of the `arg` parameter shall be as specified. For complex data elements the `arg` parameter shall be declared as `const *`. For variable-length data elements the `arg` parameter shall be followed by a length parameter of type `ioc_varlength_t`.

[VC.IOC_GEN_H.EmptyQueue]

For each configured IOC channel `Xyz` that is configured for queued communication, verify that the function or macro `IocEmptyQueue_Xyz()` calls `IOC_EmptyQueue(Xyz)`.

[VC.IOC_GEN_H.ReInit]

For each configured IOC channel `Xyz` that is configured for unqueued (*last is best*) communication, verify that the function or macro `Ioc_ReInit_Xyz()` calls `IOC_ReInit(Xyz)`.

6.2. Verification criteria for Ioc_gen_libcfg.h

When you use EB tresos IOC with EB tresos Safety OS, the IOC library is independent of the configuration. You must ensure that the library is built for use with EB tresos Safety OS.

[VC.IOC_GEN_LIBCFG_H.KernelType]

Verify that the macro `IOC_LCFG_KERNEL_TYPE` is defined as IOC_MICROKERNEL.

6.3. Verification criteria for Ioc_ChCfg.h

`Ioc_ChCfg.h` contains the generated initial values for the constant data structures that define the channel configurations for the IOC library.

[VC.IOC_CHCFG_H.Nchannels]

Verify that the macro `IOC_CFG_NCHANNELS` is defined as the number of IOC channels that you have configured.

[VC.IOC_CHCFG_H.List]

Verify that the macro `IOC_CHANNEL_CONFIGURATIONS` is defined as a list of invocations of macros whose names are of the form `Ioc_Xyz_CFG`, where `Xyz` is the name of a configured IOC channel.

[VC.IOC_CHCFG_H.ListOrder]

For each configured IOC channel `Xyz`, verify that the position of the macro `Ioc_Xyz_CFG` in `IOC_CHANNEL_CONFIGURATIONS` is equal to the channel identifier `Ioc_Xzy`.

The channel identifier macros are defined in `Ioc_gen.h`. See also VC.IOC_GEN_H.ChannelId.

[VC.IOC_CHCFG_H.DeclStateRx]

For each configured IOC channel `Xyz` that you configured for queued operation, verify that there is data declaration of `extern ioc_state_t Ioc_Xyz_State_Receiver;`.

[VC.IOC_CHCFG_H.DeclStateTx]

For each configured IOC channel `Xyz`, verify that there is data declaration of `extern ioc_state_t Ioc_Xyz_State_Sender;`.

[VC.IOC_CHCFG_H.DeclBuffer]

For each configured IOC channel `Xyz`, verify that there is data declaration of `extern Ioc_Xyz_t Ioc_Xyz_Buffer[n];`.

The number of elements (`n`) shall be

- `1` for unqueued channels
- the queue length for queued channels

[VC.IOC_CHCFG_H.DeclOffsets]

For each configured IOC channel `Xyz`, verify that there is data declaration of `extern const ioc_uint32_t Ioc_Xyz_dataElemOffsets[n];`.

The number of elements (`n`) shall be the number of data elements you configured for the channel.

[VC.IOC_CHCFG_H.DeclLengths]

For each configured IOC channel `Xyz`, verify that there is data declaration of `extern const ioc_ilength_t Ioc_Xyz_dataElemLengths[n];`.

The number of elements (`n`) shall be the number of data elements you configured for the channel.

[VC.IOC_CHCFG_H.Struct]

For each configured IOC channel `Xyz`, verify that the macro `Ioc_Xyz_CFG` is defined as an initial value for an `ioc_config_s` with member values that satisfy the verification criteria VC.IOC_CHCFG_H.Struct.*.

The structure members are (in order):

- `buffer`;
- `bufferSize`;
- `readerConfig`;
- `writerConfig`;
- `writerLockTypeLocalCore`;
- `writerLockLocalCore`;
- `writerLockTypeCrossCore`;
- `writerLockCrossCore`;
- `dataElemLengths`;
- `dataElemOffsets`;
- `numQueueEntries`;
- `commSemantics`;
- `numDataElements`;
- `initData`;

[VC.IOC_CHCFG_H.Struct.Buffer]

For each configured IOC channel `Xyz`, verify that the `buffer` member is an array of variables of type `Ioc_Xyz_t`. The number of elements in the array shall be the queue length of the configured channel. For an unqueued channel, the array length shall be `1`.

[VC.IOC_CHCFG_H.Struct.BufferSize]

For each configured IOC channel `Xyz`, verify that the `bufferSize` member is the total size of the `buffer`. If the configuration uses the `sizeof()` operator, the object of the operator shall be the whole of the array given by the `buffer` member.

[VC.IOC_CHCFG_H.Struct.ReaderConfig]

For each configured IOC channel `Xyz`, verify that the `readerConfig` member is `Ioc_Xyz_READER_END`, where `Ioc_Xyz_READER_END` is a macro that is defined as an initial value for an `ioc_channelendconfig_s` with member values that satisfy the verification criteria VC.IOC_CHCFG_H.Struct.ReaderConfig.*. The structure members are (in order):

- `trapping`;
- `accessors`;
- `state`;

[VC.IOC_CHCFG_H.Struct.ReaderConfig.Trapping]

For each configured IOC channel `Xyz`, verify that the `trapping` member of the `Ioc_Xyz_READER_END` structure is `IOC_FALSE`.

[VC.IOC_CHCFG_H.Struct.ReaderConfig.Accessors]

For each configured IOC channel `Xyz`, verify that the `accessors` member of the `Ioc_Xyz_READER_END` structure is a bitwise OR of the access bits of the OS-Applications that are allowed to read the channel.

[VC.IOC_CHCFG_H.Struct.ReaderConfig.State]

For each configured IOC channel `Xyz`, verify that the `state` member of the `Ioc_Xyz_READER_END` structure is one of:

- `IOC_NULL` if the channel is configured as unqueued.
- `&Ioc_Xyz_State_Receiver` if the channel is configured as queued.

[VC.IOC_CHCFG_H.Struct.WriterConfig]

For each configured IOC channel `Xyz`, verify that the `writerConfig` member is `Ioc_Xyz_WRITER_END`, where `Ioc_Xyz_WRITER_END` is a macro that is defined as an initial value for an `ioc_channelendconfig_s` with member values that satisfy the verification criteria VC.IOC_CHCFG_H.Struct.WriterConfig.*. The structure members are (in order):

- `trapping`;
- `accessors`;
- `state`;

[VC.IOC_CHCFG_H.Struct.WriterConfig.Trapping]

For each configured IOC channel `Xyz`, verify that the `trapping` member of the `Ioc_Xyz_WRITER_END` structure is one of:

- `IOC_FALSE` if the channel is configured for direct writing
- `IOC_TRUE` if the channel is configured for writing via trapping

[VC.IOC_CHCFG_H.Struct.WriterConfig.Accessors]

For each configured IOC channel `Xyz`, verify that the `accessors` member of the `Ioc_Xyz_WRITER_END` structure is a bitwise OR of the access bits of the OS-Applications that are allowed to write into the channel.

[VC.IOC_CHCFG_H.Struct.WriterConfig.State]

For each configured IOC channel `Xyz`, verify that the `state` member of the `Ioc_Xyz_WRITER_END` structure is `&Ioc_Xyz_State_Sender`.

[VC.IOC_CHCFG_H.Struct.WriterLockLocal.Kernel]

For each configured IOC channel `Xyz` whose `writerConfig` has `trapping=IOC_TRUE` (see VC.IOC_CHCFG_H.Struct.WriterConfig.Trapping), verify that the `writerLockTypeLocalCore` member is `IOC_LOCKINIT_NO_LOCK`.

Notes:

When IOC runs inside the microkernel, interrupts are fully locked, so an internal lock method is unnecessary. Calling an external lock method from inside the microkernel causes an immediate panic.

The `writerLockLocalCore` structure member is not used and therefore has no verification criterion.

[VC.IOC_CHCFG_H.Struct.WriterLockLocal.Thread]

For each configured IOC channel `Xyz` whose `writerConfig` has `trapping=IOC_FALSE` (see VC.IOC_CHCFG_H.Struct.WriterConfig.Trapping), verify that the `writerLockTypeLocalCore` member is:

- `IOC_LOCKINIT_NO_LOCK` if you configured the channel without locking.
- `IOC_LOCKINIT_INTRA_INT_USER` if you configured the channel with locking.

Note:

A writer lock may be needed if a channel can be written by multiple threads, regardless of whether these threads are part of the same partition or of different partitions. You are responsible for configuring the correct locking type for the channel.

The `writerLockLocalCore` structure member is not used for either of these lock methods and therefore has no verification criterion.

[VC.IOC_CHCFG_H.Struct.WriterLockXcore.Kernel]

For each configured IOC channel `Xyz` whose `writerConfig` has `trapping=IOC_TRUE` (see VC.IOC_CHCFG_H.Struct.WriterConfig.Trapping), verify that the `writerLockTypeCrossCore` member is:

- `IOC_LOCKINIT_NO_LOCK` if the channel is written from a single core.
- `IOC_LOCKINIT_INTER_KERNEL` if the channel can be written from two or more cores.

If the value of `writerLockTypeCrossCore` is `IOC_LOCKINIT_NO_LOCK`, the structure member `writerLockCrossCore` is not used and therefore has no verification criterion.

[VC.IOC_CHCFG_H.Struct.WriterLockXcore.Kernel.Lock]

For each configured IOC channel `Xyz` whose `writerLockTypeCrossCore` is `IOC_LOCKINIT_INTER_KERNEL` (see VC.IOC_CHCFG_H.Struct.WriterLockXcore.Kernel), verify that the `writerLockCrossCore` member is a numerical value that is the identifier of a spinlock in the microkernel's configuration.

[VC.IOC_CHCFG_H.Struct.WriterLockXcore.Thread]

For each configured IOC channel `Xyz` whose `writerConfig` has `trapping=IOC_FALSE` (see VC.IOC_CHCFG_H.Struct.WriterConfig.Trapping), verify that the `writerLockTypeCrossCore` member is:

- `IOC_LOCKINIT_NO_LOCK` if you configured the channel without locking.
- `IOC_LOCKINIT_INTER_USER` if you configured the channel with locking.

Note:

A writer lock may be needed if a channel can be written by different cores. You are responsible for configuring the correct locking type for the channel.

[VC.IOC_CHCFG_H.Struct.DataElemLengths]

For each configured IOC channel `Xyz`, verify that the `dataElemLengths` member is `Ioc_Xyz_dataElemLengths`.

[VC.IOC_CHCFG_H.Struct.DataElemOffsets]

For each configured IOC channel `Xyz`, verify that the `dataElemOffsets` member is `Ioc_Xyz_dataElemOffsets`.

[VC.IOC_CHCFG_H.Struct.NumQueueEntries]

For each configured IOC channel `Xyz`, verify that the `numQueueEntries` member is:

- `1`, if you configured the channel for unqueued operation.
- The queue length, if you configured the channel for queued operation.

[VC.IOC_CHCFG_H.Struct.CommSemantics]

For each configured IOC channel `Xyz`, verify that the `commSemantics` member is:

- `IOC_UNQUEUED`, if you configured the channel for unqueued communication of a single fixed-length data element.
- `IOC_UNQUEUED_EXT`, if you configured the channel for unqueued communication of a single variable-length element.
- `IOC_UNQUEUED_EXT`, if you configured the channel for unqueued communication of multiple data elements.
- `IOC_QUEUED`, if you configured the channel for queued communication of a single fixed-length data element.
- `IOC_QUEUED_EXT`, if you configured the channel for queued communication of a single variable-length element.
- `IOC_QUEUED_EXT`, if you configured the channel for queued communication of multiple data elements.

[VC.IOC_CHCFG_H.Struct.NumDataElements]

For each configured IOC channel `Xyz`, verify that the `numDataElements` member is the number of data elements that you configured for the channel.

[VC.IOC_CHCFG_H.Struct.InitData]

For each configured IOC channel `Xyz`, verify that the `initData` member is:

- `IOC_NULL` if the channel is a queued channel.
- `IOC_NULL` if the channel is an unqueued channel that you configured with a variable-length data element.
- `IOC_NULL` if the channel is an unqueued channel that you configured with a multiple data elements.
- `IOC_NULL` if the channel is an unqueued channel for which you did not configure an initial value.
- `&Ioc_Xyz_InitData` if the channel is an unqueued channel that you configured for communication of a single fixed-length data element for which you configured an initial value.

[VC.IOC_CHCFG_H.ChannelOffsets]

For each configured IOC channel `Xyz`, verify that there is a macro called `Ioc_Xyz_CHANNEL_OFFSETS` that is defined as a list of invocations of `IOC_OFFSETOF(Ioc_Xyz_t, Ioc_Xyz_DE_<n>)`.
`n` is a sequence number starting with `1`.
The number of invocations shall be the number of data elements that you configured for the channel.

When you configure IOC channels for writing by means of a trap into the microkernel, (`trapping=IOC_TRUE`), the microkernel reads data from the specified variables on behalf of the calling thread. If the caller provides a data pointer that lies in a region of memory to which the microkernel does not have read access, or that does not exist, the attempt to read the data will cause an exception inside the microkernel, which in turn leads to a shutdown.

To avoid the possibility that a failing writer can cause the entire system to shut down or cause an unexpected side effect by reading a hardware register, the configuration macro IOC_CFG_READABLEMKREGIONS defines a set of readable memory areas, one for each OS-Application. The size of this set is defined by the macro IOC_CFG_NREADABLEMKREGIONS. The microkernel verifies that each data element provided by the caller is stored entirely within the caller's readable region.

The readable region for an OS-Application can be empty. This prevents the OS-Application from writing to a trapping channel. If the OS-Application does not use trapping channels, defining its readable region as IOC_EMPTY_ADDRESSRANGE_INIT avoids the need to create an address range in the linker script.

The readable regions array has the following verification criteria.

[VC.IOC_CHCFG_H.IOC_CFG_READABLEMKREGIONS]
Verify that the macro IOC_CFG_READABLEMKREGIONS is defined as a list of address ranges and that the list contains one entry for each OS-Application that you configured.

[VC.IOC_CHCFG_H.IOC_CFG_NREADABLEMKREGIONS]
Verify that the macro IOC_CFG_NREADABLEMKREGIONS indicates the size of the list IOC_CFG_READABLEMKREGIONS.

[VC.IOC_CHCFG_H.IOC_CFG_READABLEMKREGIONS.Range]
For each OS-Application whose readable region is not IOC_EMPTY_ADDRESSRANGE_INIT, verify that the entry is IOC_ADDRESSRANGE_BEGIN_END_INIT(aBegin, aEnd) where aBegin and aEnd specify the boundaries of an area of memory. The readable region consists of all memory locations whose addresses are >=aBegin and <aEnd.

[VC.IOC_CHCFG_H.IOC_CFG_READABLEMKREGIONS.ReadByMk]
For each OS-Application whose readable region is IOC_ADDRESSRANGE_BEGIN_END_INIT(aBegin, aEnd), verify that the microkernel can read every memory location inside the region using a byte access without causing an exception when running on the same core as the OS-Application.

[VC.IOC_CHCFG_H.IOC_CFG_READABLEMKREGIONS.NoSideEffect]
For each OS-Application whose readable region is IOC_ADDRESSRANGE_BEGIN_END_INIT(aBegin, aEnd), verify that reading any location in the region cannot cause a side effect.

6.3.1. CPU-family specific verification criteria for Ioc_ChCfg.h

The following sub-sections contain verification criteria which are only valid for certain CPU-families. You must only verify your configuration against those criteria which are relevant for the CPU-family of the microcontroller for which the delivery of the IOC you are using has been qualified.

6.3.1.1. TRICORE

[VC.TRICORE.IOC_CHCFG_H.Struct.WriterLockXcore.Thread.Lock]

For each configured IOC channel `Xyz` whose `writerLockTypeCrossCore` is `IOC_LOCKINIT_INTER_USER` (see VC.IOC_CHCFG_H.Struct.WriterLockXcore.Thread), verify that the `writerLockCrossCore` member is the address of a variable of type `ioc_hwspinlockvar_t` that is placed in a memory region that is shared by all OS-Applications that are writers of this channel.

6.4. Verification criteria for Ioc_BufferTypes.h

`Ioc_BufferTypes.h` defines the data types and structures that are used as buffers for IOC.

[VC.IOC_BUFFERTYPES_H.Typedef]

For each configured IOC channel `Xyz`, verify that the data type `Ioc_Xyz_t` is defined as follows:

```
typedef struct Ioc_Xyz_s Ioc_Xyz_t
```

[VC.IOC_BUFFERTYPES_H.Struct]

For each configured IOC channel `Xyz`, verify that `Ioc_Xyz_s` is a structure declaration that contains a member

```
<type> Ioc_Xyz_DE_<n>
```

for each data element that you configured in channel `Xyz`.

`<n>` is the sequence number of the data element starting at `1` and `<type>` is the data type that you configured for the data element.

[VC.IOC_BUFFERTYPES_H.Struct.Lengths]

For each configured IOC channel `Xyz` that has more than one data element, or a single data element of variable length, verify that the first structure member of `Ioc_Xyz_s` is an array `ioc_ilength_t lengths[n]` and that the length `n` of the array is equal to the number of data elements that you configured in the channel.

6.5. Verification criteria for Ioc_configuration.c

`Ioc_configuration.c` defines the arrays of lengths and offsets that are used in the channel configuration structures in `Ioc_ChCfg.h`.

[VC.IOC_CONFIGURATION_C.Offsets]

For each configured IOC channel `Xyz`, verify that there is an array

`const ioc_uint32_t Ioc_Xyz_dataElemOffsets[n]` whose length `n` is the number of data elements that you configured in channel `Xyz`.

[VC.IOC_CONFIGURATION_C.Offsets.Init]

Verify that each `Ioc_Xyz_dataElemOffsets[]` array (see VC.IOC_CONFIGURATION_C.Offsets) is initialized with the macro `Ioc_Xyz_CHANNEL_OFFSETS`.

[VC.IOC_CONFIGURATION_C.Lengths]

For each configured IOC channel `Xyz`, verify that there is an array `const ioc_ilength_t Ioc_Xyz_dataElemLengths[n]` whose length `n` is the number of data elements that you configured in channel `Xyz`.

[VC.IOC_CONFIGURATION_C.Lengths.Init]

Verify that each `Ioc_Xyz_dataElemLengths[]` array (see VC.IOC_CONFIGURATION_C.Lengths) is initialized with the sizes of the data types for the respective data elements in channel `Xyz`, in the correct order.

6.6. Verification criteria for Ioc_gen.c

`Ioc_gen.c` contains the functions for the read and write methods that were declared as prototypes in `Ioc_gen.h`.

[VC.IOC_GEN_C.WriteSimple]

For each function `IocWrite_Xyz()` that has a primitive value parameter with no length, verify that the function calls `IOC_Write(Ioc_Xyz, &(arg0))`.

[VC.IOC_GEN_C.WriteVariable]

For each function `IocWrite_Xyz()` that has a data parameter and an `lenghtArg0` parameter, verify that the function calls `IOC_WriteExt(Ioc_Xyz, elements)`, where `elements` is an array of a single `ioc_extinput_t` that is initialized with the address of the data to send and the length. If the data parameter is declared as `const *`, verify that the `&` operator is not used.

[VC.IOC_GEN_C.SendSimple]

For each function `IocSend_Xyz()` that has a primitive value parameter, verify that the function calls `IOC_Send(Ioc_Xyz, &(arg0))`.

[VC.IOC_GEN_C.SendVariable]

For each function `IocSend_Xyz()` that has a data parameter and an `lenghtArg0` parameter, verify that the function calls `IOC_SendExt(Ioc_Xyz, elements)`, where `elements` is an array of a single `ioc_extinput_t` that is initialized with the address of the data to send and the length. If the data parameter is declared as `const *`, verify that the address-of operator (`&`) is not used.

[VC.IOC_GEN_C.WriteGroup]

For each function `IocWriteGroup_Xyz()`, verify that it calls `IOC_WriteExt(Ioc_Xyz, elements)`, where the array `elements` contains the address and size of each parameter to `IocWriteGroup_Xyz()` in left-to-right order.

For data parameters that are passed by value, the address of the parameter shall be placed in `data`. For data that is passed by reference, the parameter shall be placed in `data`.

If a data parameter is followed by a lengthArg parameter, the value of the lengthArg parameter shall be used as the size of the data.

[VC.IOC_GEN_C.SendGroup]

For each function IocSendGroup_Xyz(), verify that it calls IOC_SendExt(Ioc_Xyz, elements), where the array elements contains the address and size of each parameter to IocSendGroup_Xyz() in left-to-right order.

For data parameters that are passed by value, the address of the parameter shall be placed in data. For data that is passed by reference, the parameter shall be placed in data.

If a data parameter is followed by a lengthArg parameter, the value of the lengthArg parameter shall be used as the size of the data.

[VC.IOC_GEN_C.ReadVariable]

For each function IocRead_Xyz(arg0, ioc_varlength_t *lengthArg0), verify that it calls IOC_ReadExt(Ioc_Xyz, lengths, destinations), where the lengths and destinations arrays each contain a single element. Verify that the arg0 parameter is copied to destinations[0] before calling IOC_ReadExt() and that lengths[0] is copied to the variable referenced by lengthArg0 after IOC_ReadExt() returns.

[VC.IOC_GEN_C.ReadGroup]

For each function IocReadGroup_Xyz(), verify that it calls IOC_ReadExt(Ioc_Xyz, lengths, destinations), where the array destinations has as many elements as there are argN parameters and is initialized with the values of the corresponding parameters. Verify that, after IOC_ReadExt() returns, the variable referenced by each lengthArgN parameter is written with the corresponding element of the lengths array.

Note: the elements of the lengths array that do not correspond to lengthArgN parameters are discarded.

[VC.IOC_GEN_C.ReceiveVariable]

For each function IocReceive_Xyz(arg0, ioc_varlength_t *lengthArg0), verify that it calls IOC_ReceiveExt(Ioc_Xyz, lengths, destinations), where the lengths and destinations arrays each contain a single element. Verify that the arg0 parameter is copied to destinations[0] before calling IOC_ReceiveExt() and that lengths[0] is copied to the variable referenced by lengthArg0 after IOC_ReceiveExt() returns.

[VC.IOC_GEN_C.ReceiveGroup]

For each function IocReceiveGroup_Xyz(), verify that it calls IOC_ReceiveExt(Ioc_Xyz, lengths, destinations), where the array destinations has as many elements as there are argN parameters and is initialized with the values of the corresponding parameters. Verify that, after IOC_ReceiveExt() returns, the variable referenced by each lengthArgN parameter is written with the corresponding element of the lengths array.

Note: the elements of the lengths array that do not correspond to lengthArgN parameters are discarded.

6.7. Verification criteria for Ioc_data_*.c

For every IOC channel that you configured there is a generated source file that contains the data buffer and the state information that the writer or sender needs to update. For queued channels, there is in addition a generated source file that contains the state information that the receiver needs to update. You must verify that these files are correctly generated and that your microkernel configuration places them in appropriate memory regions to which access can be granted or denied according to your safety requirements and analysis.

[VC.IOC_DATA_CHXX_SEND_C.Ioc_data_app_chNN_Send_c]

For each IOC channel `Xyz` that you configured for direct (non-trapping) access, verify that there is a file called `Ioc_data_app_chNN_Send.c`, where `NN` is the numerical channel ID generated in `Ioc_gen.h`. `NN` may contain one or more numerical digits.

[VC.IOC_DATA_CHXX_SEND_C.Ioc_data_app_chNN_Send_c.Location]

Verify that your microkernel configuration places the contents of the file (see VC.IOC_DATA_CHXX_SEND_C.Ioc_data_app_chNN_Send_c) in a memory region that can be written by the configured writers of the channel. In addition, verify that the memory region cannot be written by the microkernel, by the QM-OS, or by any other unauthorized executable object.

[VC.IOC_DATA_CHXX_SEND_C.Ioc_data_kern_cXX_chNN_Send_c]

For each IOC channel `Xyz` that you configured for trapping access with writers on a single core, verify that there is a file called `Ioc_data_kern_cXX_chNN_Send.c`, where
`XX` is the numerical core ID of the writers and
where `NN` is the numerical channel ID generated in `Ioc_gen.h`.
`XX` and `NN` may each contain one or more numerical digits.

[VC.IOC_DATA_CHXX_SEND_C.Ioc_data_kern_cXX_chNN_Send_c.Location]

Verify that your microkernel configuration places the contents of the file (see VC.IOC_DATA_CHXX_SEND_C.Ioc_data_kern_cXX_chNN_Send_c) in a memory region that can be written only by the microkernel on core `XX`.

[VC.IOC_DATA_CHXX_SEND_C.Ioc_data_kern_shared_chNN_Send_c]

For each IOC channel `Xyz` that you configured for trapping access with writers on two or more cores, verify that there is a file called `Ioc_data_kern_shared_chNN_Send.c`, where `NN` is the numerical channel ID generated in `Ioc_gen.h`. `NN` may contain one or more numerical digits.

[VC.IOC_DATA_CHXX_SEND_C.Ioc_data_kern_shared_chNN_Send_c.Location]

Verify that your microkernel configuration places the contents of the file (see VC.IOC_DATA_CHXX_SEND_C.Ioc_data_shared_chNN_Send_c) in a memory region that can be written by the microkernel on all cores on which the writers reside.

[VC.IOC_DATA_CHXX_SEND_C.Data_Buffer]

For each IOC channel `Xyz` that you configured, verify that its sender data file `Ioc_data_*_Send.c` contains an array `Ioc_Xyz_t Ioc_Xyz_Buffer[QQ]`, where:

- `QQ` is 1 if the channel is unqueued.

- QQ is the configured queue length if the channel is queued.

[VC.IOC_DATA_CHXX_SEND_C.InitData]

For each IOC channel Xyz that you configured with initial data, verify that its sender data file Ioc_data_*_Send.c contains a constant object Ioc_Xyz_InitData that has the type and initial value(s) that you configured for the channel.

[VC.IOC_DATA_CHXX_SEND_C.Data_State]

For each IOC channel Xyz that you configured, verify that its sender data file Ioc_data_*_Send.c contains an IOC state variable ioc_state_t Ioc_Xyz_State_Sender.

[VC.IOC_DATA_CHXX_RECEIVE_C.Ioc_data_app_chNN_Receive_c]

For each IOC channel Xyz that you configured for queued communication, verify that there is a file called Ioc_data_app_chNN_Receive.c, where NN is the numerical channel ID generated in Ioc_gen.h. NN may contain one or more numerical digits.

[VC.IOC_DATA_CHXX_RECEIVE_C.Ioc_data_app_chNN_Receive_c.Location]

Verify that your microkernel configuration places the contents of the file (see VC.IOC_DATA_CHXX_RECEIVE_C.Ioc_data_app_chNN_Receive_c) in a memory region that can be written by the configured receivers of the channel. In addition, verify that the memory region cannot be written by the microkernel, by the QM-OS, or by any other unauthorized executable object.

[VC.IOC_DATA_CHXX_RECEIVE_C.Data_State]

For each IOC channel Xyz that you configured for queued communication, verify that its receiver data file Ioc_data_*_Receive.c contains an IOC state variable ioc_state_t Ioc_Xyz_State_Receiver.

6.8. Verification criteria for the operating system

The freedom from interference provided by IOC depends on the spatial freedom from interference provided by the operating system. IOC requires that the operating system provides memory protection and mutual exclusion to the same ASIL as that required of IOC.

6.8.1. Configuration of memory regions

Senders and receivers are considered separately. The read API for non-queued channels requires only read access to the data buffer and sender state. The only interference that can occur is caused by a read-during-write.

The receive API for queued channels also requires read access to the data buffer and sender state. In addition it requires write access to the receiver state information. Since IOC does not support multiple receivers for queued channels, you can place the state variables into an existing data region for the receiver's executable.

For writers and senders there are two cases to consider:

- Non-trapping channels
- Trapping channels

Non-trapping channels. In the case of non-trapping channels the sender's executable needs write access to the data buffer and sender state.

- If there is a single executable that contains all writers or senders you can place the buffer and state variable into an existing data region for the executable.
- If there are multiple writers the buffer and state variable must be placed into a region that can be written by all senders. If your application has many channels with multiple writers you might be restricted by the number of regions supported in the hardware. If your requirements do not permit you sufficient freedom to aggregate channels into larger common memory regions, the only solution is to use trapping channels.

Trapping channels. In the case of trapping channels the executable does not need write access to the data buffer and sender state. Instead, the buffer and state are modified by the microkernel itself. The placement of the IOC data in this case depends on the core assignment of the writers. If all the writers are assigned to the same core, you can place the buffer and sender state into the region that contains the microkernel's own variables. If the writers are assigned to two or more cores, you must place the buffer and sender state into a memory region that is accessible by all the cores. You must open a window in the configuration of the hardware's bus MPU to permit all cores to access the memory region. You must determine whether such an arrangement satisfies your freedom from interference requirements. This is particularly important if your hardware has cores that do not fully satisfy the *reliable execution environment* assumption made by the IOC's design. If you are using such hardware you must separate your cross-core channels into high-integrity and low-integrity regions. You can permit all cores to access the low-integrity region, but you must not allow the unreliable cores to have write access to the high-integrity region. If your hardware has a non-uniform memory architecture and does not provide shared memory with equal arbitration for all cores, you must choose the placement of the shared regions carefully, to avoid potential performance problems caused by frequent cross-core memory accesses.

The above guidelines result in the following verification criteria, but you must bear in mind that these criteria are not exhaustive. You must configure your system carefully depending on your own requirements.

[VC.IOC_MemoryProtection.FFI_Req]
Verify that the configuration of memory regions for IOC satisfies your freedom from interference requirements.

[VC.IOC_MemoryProtection.BusMpuNonLockstep]
Verify that your configuration of the bus MPU does not permit unreliable cores to write to IOC channels that must be free from interference.

[VC.IOC_MemoryProtection.BusMpuMicrokernel]
Verify that your configuration of the bus MPU does not permit a core to write to the microkernel's variables or stack that belongs to another core.

6.8.2. Configuration of mutual exclusion

To prevent multiple senders from updating the channel data concurrently, and to prevent a reader from interrupting a writer on the same core, you must configure appropriate locks in the operating system and assign the locks to the channels.

WARNING ⚠

Possible deadlocks

The reader is blocked if the writer gets preempted during a write operation until the writer resumes its operation. This may result in deadlocks, e.g. if the reader resides on the same core as the writer and preempts the writer during the write operation.

You must also ensure that the channel initialization API (`IocReinit_Xyz()` and `IocEmptyQueue_Xyz()`) cannot execute concurrently with any read or write operations on the same channel. These APIs do not use the configured locks, so you must implement your own mutual exclusion.

When you configure a cross-core lock (spinlock) for use with non-trapping channels, always configure the spinlock to lock interrupts at the highest level (the level of SuspendAllInterrupts) to avoid a preemption on the locking core from locking out other cores for an extended duration.

[VC.IOC_Locks.ReaderInterrruptsWriter]

For each non-trapping channel whose readers can interrupt its writers on the same core, verify that you have configured a lock that prevents any reader from interrupting a writer.

[VC.IOC_Locks.WriterInterrruptsWriter]

For each non-trapping channel with multiple writers, verify that you have configured a lock that prevents any writer from interrupting any other writer and from executing concurrently with any writer on a different core.

[VC.IOC_Locks.ReceiverInterruptsReceiver]

Verify that you do not use the `IocReceive_Xyz()` and `IocReceiveGroup_Xyz()` APIs from more than one task or ISR.

[VC.IOC_Locks.CombinedLock]

For every OS spinlock object that you configure for use with IOC, verify that you configure the lock to block interrupts at the `SuspendAllInterrupts()` level.

[VC.IOC_Locks.SpinlockInterruptLock]

For every thread-level spinlock that you configure for use with IOC, verify that you configure the channel with a core-local lock that blocks at least at the level of `SuspendAllInterrupts()`.

[VC.IOC_Locks.InitializeApi]

If your application calls `IocReinit_Xyz()` or `IocEmptyQueue_Xzy()` verify that no read or write access to channel `Xyz` can occur concurrently.

6.9. Other considerations

6.9.1. Reliance on operating system

EB tresos IOC relies on features of the operating system to provide freedom from interference for the participants in a communication channel. If the operating system does not have the required integrity level then neither does EB tresos IOC. If the operating system does not provide memory protection and mutual exclusion to the required integrity level, then EB tresos IOC's integrity level is correspondingly lower.

EB tresos IOC has only been qualified for use with the version of EB tresos Safety OS with which it was delivered.

EB tresos IOC attaches itself to EB tresos Safety OS by means of the "addon" feature. This means that the verification criterion `[VC.Mk_gen_config.global.25]` in the EB tresos Safety OS safety manual [SAFETYMAN] is no longer in force.

[VC.IOC_OsDependence]
Verify that you use EB tresos IOC with EB tresos Safety OS.

[VC.IOC_Mk_gen_config.MK_nAddOns]
In the generated file `Mk_gen_config.h`, verify that the value of the macro `MK_CFG_NADDONS` is equal to the number of addon modules that you configured for use.
This verification criterion replaces `[VC.Mk_gen_config.global.25]` from the EB tresos Safety OS safety manual.

[VC.IOC_Mk_gen_config.MK_addOnTable]
In the generated file `Mk_gen_addon.h`, verify that the value of the macro `MK_CFG_ADDONCONFIG` is defined as a comma-separated list of addon descriptors for the addon modules that you configured for use.

[VC.IOC_Mk_gen_config.MK_addOnTable.IocEntry]
In the generated file `Mk_gen_addon.h`, verify that the macro `MK_CFG_ADDONCONFIG` contains an entry for IOC. The entry shall be `&IOC_mkaddon_descriptor`.

[VC.IOC_Mk_gen_user.IocAddOnId]
In the generated file `Mk_gen_user.h`, verify that the macro `MK_ADDON_ID_IOC` is defined with a value equal to the index of the IOC descriptor in `MK_CFG_ADDONCONFIG`.
(See `VC.IOC_Mk_gen_config.MK_addOnTable.IocEntry`.)

6.9.2. Representation of data

You must ensure that every participant in an IOC communication understands the content of information in the same way. A different view of data can occur under the following circumstances:

- Two communicating cores have different word length.
- Two communicating cores have different endianness.
- Two communicating cores have different floating point hardware.
- Two communicating software components use different compiler options that affect structure padding.
- Two communicating software components use different compiler options that affect the size of enumerated types.
- Two communicating software components use different compiler options that affect floating point precision.

This list is not exhaustive. It is your responsibility to understand the factors that can cause differences in representation and to avoid them.

[VC.IOC_DataRepresentation]
For each configured channel, verify that all participants in the channel have the same representation of the data that is transferred through the channel.

[VC.IOC_DataElementSize]
For every complex data type that you used in the configuration of a data element, verify that its size is strictly less than 65536 bytes.

6.9.3. Initial values

The initial value of the buffer in an unqueued channel might not be defined. In the case of a channel with a single fixed-length data element, it is possible to configure an initial value. However, if the channel has a variable-length element or has multiple data elements, the initial state of the buffer is undefined. Furthermore, the length for variable-length elements is initialized to the configured maximum length.

At startup, the default initialization of C variables fills the buffer with zero. However, after a shutdown, IOC does not re-initialize the buffer, so it retains its previous value, even though IOC initializes the length to the maximum configured value.

You must ensure that your application cannot be adversely affected by the initial state of unqueued channels. Alternatively, ensure that each such channel is written before reading.

[VC.IOC_UnqueuedInitialState]
Verify that your application cannot be adversely affected by an undefined initial value in an unqueued channel.

6.9.4. Recovery from faults

If your application is designed to be resilient to protection faults and can request the termination of a task, an ISR or an OS-Application by means of the return value from the `ProtectionHook()` or explicitly by an API call, you must consider that the protection fault or the termination could have occurred during read or write operations on IOC channels. An affected channel could be in an incorrect state.

You may need to reinitialize all IOC channels that could be affected by the task or application that caused the protection fault.

It is not possible for this document to give specific criteria for cleaning up after a forced termination. You must analyze your system and determine what effect corrupt data could have, including an endless wait if a reader is waiting for an update of a channel to complete. Therefore there is a single verification criterion.

[VC.IOC_Cleanup]

Verify that your configuration and use of EB tresos IOC is robust against any possible forced termination of a task, ISR, or OS-Application that uses IOC.

6.9.5. Concurrent changes to IOC arguments

Among the parameters of extended IOC API functions are arrays, which effectively are located in memory. Since IOC internally accesses these memory locations directly during its operation, it is vitally important that these parameters are not changed while IOC is operating on them, or else undefined behaviour can occur.

[VC.IOC_StableArguments.Sender]

Verify that the content of `ioc_extinput_t` arguments to `IOC_WriteExt` and `IOC_SendExt` is not changed while IOC processes these calls.

The following aspects help you to verify this criterion:

- When you use the IOC functions generated by the *OS generator*, the input data resides on the stack. Usually no thread should be allowed to write to another thread's stack.
- The processing of trapping APIs cannot be interrupted by other executable entities. Therefore, concurrent modifications of arguments to trapping APIs can only be caused by other cores or other bus masters.

6.9.6. Memory mapping of IOC channels

You must ensure that all IOC channels are mapped to a memory type where `IOC_HwMemoryBarrier()` can ensure memory-access ordering. `IOC_HwMemoryBarrier()` usually is implemented as the strongest synchronization instruction or instruction sequence that is available on the target hardware. Nevertheless, dependent on your system, there may be memory to which it can't be ensured that read or write access is completely finished after the return of `IOC_HwMemoryBarrier()`. An IOC channel must not be mapped to such a memory type.

[VC.IOC_Memory.barrier]

Verify that all IOC channels are mapped to a memory type where `IOC_HwMemoryBarrier()` can ensure memory-access ordering.

Note:

You can find the implemenation of `IOC_HwMemoryBarrier()` in `include/private/<CPUFAMILY>/Ioc_<CPUFAMILY>_core.h`. You must evaluate the implementation carefully and check to which type of memory it applies.

Appendix A. Reserved symbols

Identifiers that begin with the sequence `IOC_` in any combination of upper- and lower-case are reserved for use in EB tresos IOC. Do not define any objects whose names begin with this prefix in your software components or in your configuration.

The symbols beginning with the prefixes in the following list are defined by the AUTOSAR OS specification. Do not define any objects whose names begin with any of these prefixes in your software components or in your configuration.

- IocRead
- IocReadGroup
- IocWrite
- IocWriteGroup
- IocSend
- IocSendGroup
- IocReceive
- IocReceiveGroup
- Ioc_ReInit
- IocEmptyQueue

Appendix B. Safety of services

The entire API of EB tresos IOC is designed to be used by software components of any safety integrity level up to ASIL-D, provided that the verification criteria in Chapter 6, “Configuration verification criteria“ are fully satisfied.

However, EB tresos IOC provides no guarantee about the content of the data that is transferred. If a software component provides corrupt data to a write or send method, the corrupt data is transferred to the receiver.

In the case of a channel whose writers operate in non-trapping mode, all configured writers can write directly into the data buffer and sender state of the channel without calling the EB tresos IOC API. The writer can thus corrupt not only the data but also the state information. At worst, this behavior only affects the content transferred to the reader in a manner similar to corrupt data being written by means of the API. In cases of extreme corruption (for example, variable-length data with a length longer than configured) the read API reports an error.

Appendix C. Document configuration information

This document has been created by the DocBook engine using the source files and revisions listed below. All paths are relative to the directory https://subversion.ebgroup.elektrobit.com/svn/autosar/asc_Ioc/branches/ASCMICROOS-4772_inspection_TRICORE/doc/public/safety_manual.

Filename	Revision
../../project/fragments/cross_references/Cross_References.xml	2875
../../project/fragments/glossary/Glossary.xml	1360
../fragments/entities/IOC.ent.m4	1773
../fragments/top_level_requirements/Assumptions.xml	1763
../fragments/top_level_requirements/Definitions.xml	1678
../fragments/top_level_requirements/Level1.xml	1359
../fragments/top_level_requirements/Level2.xml	1503
../fragments/top_level_requirements/Overview.xml	583
../fragments/top_level_requirements/SEooC.xml	1481
8.6_IOC_safety_manual.xml	1373
SM_ApiSafety.xml	1360
SM_Assumed_Requirements.xml	1335
SM_ConfigCriteria.xml	2885
SM_Description.xml	1746
SM_Document_information.xml	1369
SM_History.xml	2889
SM_ReservedWords.xml	1335
SM_SafeUse.xml	1369

Glossary

`ECC` Error (detection) and correction codes are redundant information stored next to the data itself so that you can detect and eventually also correct errors.

`lockstep mode` Lockstep mode means that hardware components are replicated and perform the identical function for each clock cycle. The output of the hardware components is compared. If the output diverges, the error is detected and appropriate hardware-specific actions are performed.

`OS generator` The OS generator creates C-source code from the OS configuration in EB tresos Studio.

Bibliography

[IOCSMINSP1] *Inspection Report 1 of the EB tresos IOC Safety Manual*

[IOCSMINSP2] *Inspection Report 2 of the EB tresos IOC Safety Manual*

[IOCSMINSP3] *Inspection Report 3 of the EB tresos IOC Safety Manual*

[IOCSMINSP4] *Inspection Report 4 of the EB tresos IOC Safety Manual*

[IOCSMINSP5] *Inspection Report 5 of the EB tresos IOC Safety Manual*

[IOCUSRDOC] *EB tresos® IOC User's Guide*

3.5_IOC_documentation.pdf

[ISO/IEC 61508:2010] *Functional safety of electrical/electronic/programmable electronic safety-related systems*, 2010

[ISO/IEC 61508-3:2010] *Functional safety of electrical/electronic/programmable electronic safety-related systems*: Part 3: Software requirements, 2010

[ISO24765] *ISO/IEC/IEE 24765 First edition*: Systems and software engineering - Vocabulary, 2010

[ISO26262_1ST] *INTERNATIONAL STANDARD ISO 26262*: Road vehicles - Functional safety, 2011

[ISO26262-1_1ST] *INTERNATIONAL STANDARD ISO 26262-1*: Road vehicles - Functional safety - Part 1: Vocabulary, 2011

[ISO26262-10_1ST] *INTERNATIONAL STANDARD ISO 26262-10*: Road vehicles - Functional safety - Part 10: Guideline on ISO 26262, 2012

[ISO26262-3_1ST] *INTERNATIONAL STANDARD ISO 26262-3*: Road vehicles - Functional safety - Part 3: Concept phase, 2011

[ISO26262-6_1ST] *INTERNATIONAL STANDARD ISO 26262-6*: Road vehicles - Functional safety - Part 6: Product development at the software level, 2011

[KNOWN-PROBLEMS] *EB tresos AutoCore known problems (EB_tresos_Autocore_known_issues-SAFETY-OS_<version>_COMMON_SRC.pdf)*

[Q-STATEMENT] *EB tresos AutoCore Quality Statement Safety OS*

(EB_tresos_Autocore-quality_statement-SafetyOs_<version>_<derivative>.pdf)

[SAFETYMAN] *EB tresos Safety OS safety manual*

8.2_Safety_OS_safety_manual_<processor_family>.pdf