

RTA-OS TraveoII/GHS V5.0.2

Port Guide

Status: Released



Copyright

The data in this document may not be altered or amended without special notification from ETAS GmbH. ETAS GmbH undertakes no further obligation in relation to this document. The software described in it can only be used if the customer is in possession of a general license agreement or single license. Using and copying is only allowed in concurrence with the specifications stipulated in the contract. Under no circumstances may any part of this document be copied, reproduced, transmitted, stored in a retrieval system or translated into another language without the express written permission of ETAS GmbH.

© Copyright 2008-2021 ETAS GmbH, Stuttgart.

The names and designations used in this document are trademarks or brands belonging to the respective owners.

Document: 10793-PG-5.0.2 EN-02-2021

Safety Notice

This ETAS product fulfills standard quality management requirements. If requirements of specific safety standards (e.g. IEC 61508, ISO 26262) need to be fulfilled, these requirements must be explicitly defined and ordered by the customer. Before use of the product, customer must verify the compliance with specific safety standards.

Contents

1	Introduction	7
1.1	About You	7
1.2	Document Conventions	8
1.3	References	8
2	Installing the RTA-OS Port Plug-in	9
2.1	Preparing to Install	9
2.1.1	Hardware Requirements	9
2.1.2	Software Requirements	9
2.2	Installation	10
2.2.1	Installation Directory	10
2.3	Licensing	11
2.3.1	Installing the ETAS License Manager	11
2.3.2	Licenses	12
2.3.3	Installing a Concurrent License Server	13
2.3.4	Using the ETAS License Manager	14
2.3.5	Troubleshooting Licenses	16
3	Verifying your Installation	19
3.1	Checking the Port	19
3.2	Running the Sample Applications	19
4	Port Characteristics	21
4.1	Parameters of Implementation	21
4.2	Configuration Parameters	21
4.2.1	Stack used for C-startup	21
4.2.2	Stack used when idle	22
4.2.3	Stack overheads for ISR activation	22
4.2.4	Stack overheads for ECC tasks	22
4.2.5	Stack overheads for ISR	22
4.2.6	Orti22/Lauterbach	23
4.2.7	Orti23/Lauterbach	23
4.2.8	ORTI Stack Fill	23
4.2.9	Block default interrupt	24
4.2.10	Enable stack repositioning	24
4.2.11	GetAbortStack always	24
4.2.12	Alignment boundary setting	25
4.2.13	Link Type	25
4.2.14	Set floating-point mode	25
4.2.15	Set floating-point support	25
4.2.16	Stack protector setting	26
4.2.17	Hide spurious interrupt	26
4.2.18	Clear demux isrs when lowering ipl	26
4.2.19	CrossCore0 IPC	27
4.2.20	CrossCore1 IPC	27
4.2.21	SpinLock IPC	27
4.2.22	Use Default VTOR	28

4.3	Generated Files	28
5	Port-Specific API	30
5.1	API Calls	30
5.1.1	Os_AwaitStartup	30
5.1.2	Os_InitializeVectorTable	31
5.2	Callbacks	32
5.2.1	Os_Cbk_GetAbortStack	32
5.2.2	Os_Cbk_GetSetProtection	33
5.2.3	Os_Cbk_SetMemoryAccess	34
5.2.4	Os_Cbk_StartCore	38
5.3	Macros	39
5.3.1	CAT1_ISR	39
5.3.2	Os_ClearDemuxInterrupts	40
5.3.3	Os_Clear_x	40
5.3.4	Os_DisableAllConfiguredInterrupts	40
5.3.5	Os_Disable_x	41
5.3.6	Os_EnableAllConfiguredInterrupts	41
5.3.7	Os_Enable_x	41
5.3.8	Os_IntChannel_x	42
5.4	Type Definitions	42
5.4.1	Os_StackSizeType	42
5.4.2	Os_StackValueType	42
6	Toolchain	43
6.1	Compiler Versions	43
6.1.1	Green Hills Software, Compiler v2017.1.4	43
6.1.2	Green Hills Software, Compiler v2019.1.4	43
6.2	Options used to generate this guide	44
6.2.1	Compiler	44
6.2.2	Assembler	45
6.2.3	Librarian	46
6.2.4	Linker	46
6.2.5	Debugger	47

7	Hardware	49
7.1	Supported Devices	49
7.2	Register Usage	50
	7.2.1 Initialization	50
	7.2.2 Modification	51
7.3	Required OS resources	52
7.4	Interrupts	52
	7.4.1 Interrupt Priority Levels	53
	7.4.2 Allocation of ISRs to Interrupt Vectors	53
	7.4.3 Vector Table	54
	7.4.4 Writing Category 1 Interrupt Handlers	54
	7.4.5 Writing Category 2 Interrupt Handlers	55
	7.4.6 Category 2 Software Interrupts	55
	7.4.7 Default Interrupt	55
7.5	Memory Model	55
7.6	Processor Modes	55
7.7	Stack Handling	56
7.8	Processor state when calling StartOS()	56
7.9	Spurious interrupts when termination an application	56
7.10	Use of the Inter-Processor Communication hardware in multicore applications	56
8	Performance	57
8.1	Measurement Environment	57
8.2	RAM and ROM Usage for OS Objects	57
8.3	Stack Usage	58
8.4	Library Module Sizes	58
8.5	Execution Time	61
	8.5.1 Context Switching Time	62
9	Finding Out More	64
10	Contacting ETAS	65
10.1	Technical Support	65
10.2	General Enquiries	65
	10.2.1 ETAS Global Headquarters	65
	10.2.2 ETAS Local Sales & Support Offices	65

1 Introduction

RTA-OS is a small and fast real-time operating system that conforms to both the AUTOSAR OS (R3.0.1 -> R3.0.7, R3.1.1 -> R3.1.5, R3.2.1 -> R3.2.2, R4.0.1 -> R4.5.0 (R19-11)) and OSEK/VDX 2.2.3 standards (OSEK is now standardized in ISO 17356). The operating system is configured and built on a PC, but runs on your target hardware.

This document describes the RTA-OS TraveoII/GHS port plug-in that customizes the RTA-OS development tools for the Cypress Semiconductor TraveoII with the GHS compiler. It supplements the more general information you can find in the *User Guide* and the *Reference Guide*.

The document has two parts. Chapters 2 to 3 help you understand the TraveoII/GHS port and cover:

- how to install the TraveoII/GHS port plug-in;
- how to configure TraveoII/GHS-specific attributes;
- how to build an example application to check that the TraveoII/GHS port plug-in works.

Chapters 4 to 8 provide reference information including:

- the number of OS objects supported;
- required and recommended toolchain parameters;
- how RTA-OS interacts with the TraveoII, including required register settings, memory models and interrupt handling;
- memory consumption for each OS object;
- memory consumption of each API call;
- execution times for each API call.

For the best experience with RTA-OS it is essential that you read and understand this document.

1.1 About You

You are a trained embedded systems developer who wants to build real-time applications using a preemptive operating system. You should have knowledge of the C programming language, including the compilation, assembling and linking of C code for embedded applications with your chosen toolchain. Elementary knowledge about your target microcontroller, such as the start address, memory layout, location of peripherals and so on, is essential.

You should also be familiar with common use of the Microsoft Windows operating system, including installing software, selecting menu items, clicking buttons, navigating files and folders.

1.2 Document Conventions

The following conventions are used in this guide:

- | | |
|------------------------------------|--|
| Choose File > Open . | Menu options appear in bold, blue characters. |
| Click OK . | Button labels appear in bold characters |
| Press <Enter>. | Key commands are enclosed in angle brackets. |
| The "Open file" dialog box appears | GUI element names, for example window titles, fields, etc. are enclosed in double quotes. |
| Activate(Task1) | Program code, header file names, C type names, C functions and API call names all appear in a monospaced typeface. |
| See Section 1.2. | Internal document hyperlinks are shown in blue letters . |



Functionality in RTA-OS that might not be portable to other implementations of AUTOSAR OS is marked with the RTA-OS icon.



Important instructions that you must follow carefully to ensure RTA-OS works as expected are marked with a caution sign.

1.3 References

OSEK is a European automotive industry standards effort to produce open systems interfaces for vehicle electronics. OSEK is now standardized in ISO 17356. For details of the OSEK standards, please refer to:

<https://www.iso.org/standard/40079.html>

AUTOSAR (AUTomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers. For details of the AUTOSAR standards, please refer to:

<http://www.autosar.org>

2 Installing the RTA-OS Port Plug-in

2.1 Preparing to Install

RTA-OS port plug-ins are supplied as a downloadable electronic installation image which you obtain from the ETAS Web Portal. You will have been provided with access to the download when you bought the port. You may optionally have requested an installation CD which will have been shipped to you. In either case, the electronic image and the installation CD contain identical content.



Integration Guidance 2.1: *You must have installed the RTA-OS tools before installing the Traveoll/GHS port plug-in. If you have not yet done this then please follow the instructions in the Getting Started Guide.*

2.1.1 Hardware Requirements

You should make sure that you are using at least the following hardware before installing and using RTA-OS on a host PC:

- 1GHz Pentium Windows-capable PC.
- 2G RAM.
- 20G hard disk space.
- CD-ROM or DVD drive (Optional)
- Ethernet card.

2.1.2 Software Requirements

RTA-OS requires that your host PC has one of the following versions of Microsoft Windows installed:

- Windows 8
- Windows 10



Integration Guidance 2.2: *The tools provided with RTA-OS require Microsoft's .NET Framework v2.0 (included as part of .NET Framework v3.5) and v4.5.2 to be installed. You should ensure that these have been installed before installing RTA-OS. The .NET framework is not supplied with RTA-OS but is freely available from <https://www.microsoft.com/net/download>. To install .NET 3.5 on Windows 10 see <https://docs.microsoft.com/en-us/dotnet/framework/install/dotnet-35-windows-10>.*

The migration of the code from v2.0 to v4.x will occur over a period of time for performance and maintenance reasons.

2.2 Installation

Target port plug-ins are installed in the same way as the tools:

1. Either

- Double click the executable image; or
- Insert the RTA-OS TraveoII/GHS CD into your CD-ROM or DVD drive.

If the installation program does not run automatically then you will need to start the installation manually. Navigate to the root directory of your CD/DVD drive and double click `autostart.exe` to start the setup.

2. Follow the on-screen instructions to install the TraveoII/GHS port plug-in.

By default, ports are installed into `C:\ETAS\RTA-OS\Targets`. During the installation process, you will be given the option to change the folder to which RTA-OS ports are installed. You will normally want to ensure that you install the port plug-in in the same location that you have installed the RTA-OS tools. You can install different versions of the tools/targets into different directories and they will not interfere with each other.



Integration Guidance 2.3: *Port plug-ins can be installed into any location, but using a non-default directory requires the use of the `--target_include` argument to both `rtaosgen` and `rtaoscfg`. For example:*

```
rtaosgen --target_include:<target_directory>
```

2.2.1 Installation Directory

The installation will create a sub-directory under `Targets` with the name `TraveoIIIGHS_5.0.2`. This contains everything to do with the port plug-in.

Each version of the port installs in its own directory - the trailing `_5.0.2` is the port's version identifier. You can have multiple different versions of the same port installed at the same time and select a specific version in a project's configuration.

The port directory contains:

TraveoIIIGHS.dll - the port plug-in that is used by `rtaosgen` and `rtaoscfg`.

RTA-OS TraveoIIIGHS Port Guide.pdf - the documentation for the port (the document you are reading now).

RTA-OS TraveoIIIGHS Release Note.pdf - the release note for the port. This document provides information about the port plug-in release, including a list of changes from previous releases and a list of known limitations.

There may be other port-specific documentation supplied which you can also find in the root directory of the port installation. All user documentation is distributed in PDF format which can be read using Adobe Acrobat Reader. Adobe Acrobat Reader is not supplied with RTA-OS but is freely available from <http://www.adobe.com>.

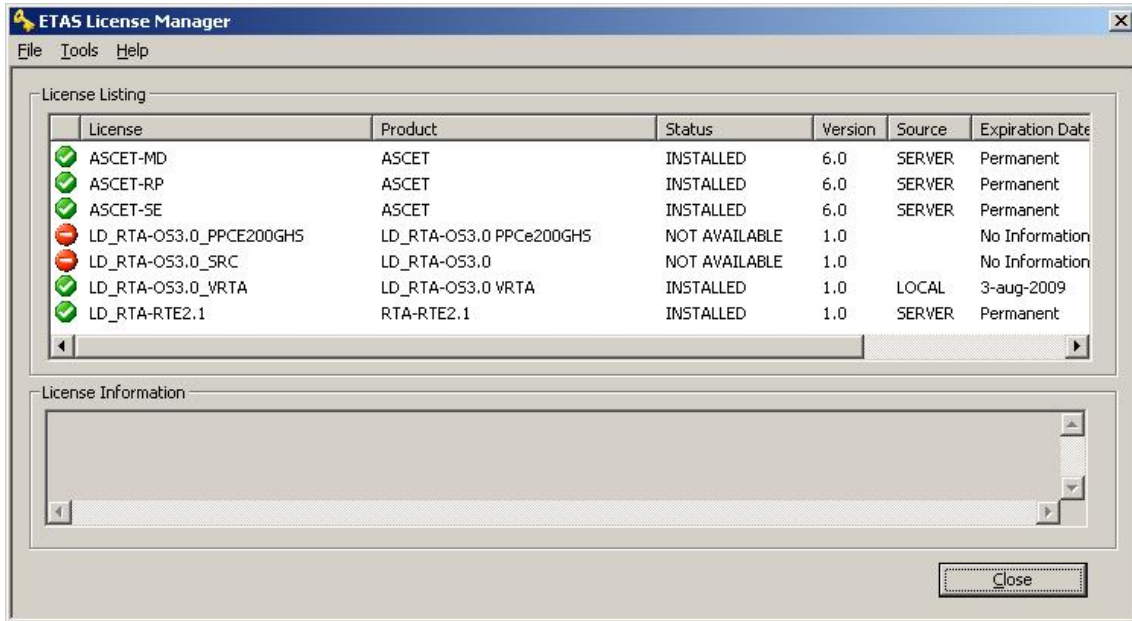


Figure 2.1: The ETAS License manager

2.3 Licensing

RTA-OS is protected by FLEXnet licensing technology. You will need a valid license key in order to use RTA-OS.

Licenses for the product are managed using the ETAS License Manager which keeps track of which licenses are installed and where to find them. The information about which features are required for RTA-OS and any port plug-ins is stored as license signature files that are stored in the folder <install_folder>\bin\Licenses.

The ETAS License Manager can also tell you key information about your licenses including:

- Which ETAS products are installed
- Which license features are required to use each product
- Which licenses are installed
- When licenses expire
- Whether you are using a local or a server-based license

Figure 2.1 shows the ETAS License Manager in operation.

2.3.1 Installing the ETAS License Manager



Integration Guidance 2.4: *The ETAS License Manager must be installed for RTA-OS to work. It is highly recommended that you install the ETAS License Manager during your installation of RTA-OS.*

The installer for the ETAS License Manager contains two components:

1. the ETAS License Manager itself;
2. a set of re-distributable FLEXnet utilities. The utilities include the software and instructions required to setup and run a FLEXnet license server manager if concurrent licenses are required (see Sections 2.3.2 and 2.3.3 for further details)

During the installation of RTA-OS you will be asked if you want to install the ETAS License Manager. If not, you can install it manually at a later time by running `<install_folder>\LicenseManager\LicensingStandaloneInstallation.exe`.

Once the installation is complete, the ETAS License Manager can be found in `C:\Program Files\Common Files\ETAS\Licensing`.

After it is installed, a link to the ETAS License Manager can be found in the Windows Start menu under **Programs → ETAS → License Management → ETAS License Manager**.

2.3.2 Licenses

When you install RTA-OS for the first time the ETAS License Manager will allow the software to be used in *grace mode* for 14 days. Once the grace mode period has expired, a license key must be installed. If a license key is not available, please contact your local ETAS sales representative. Contact details can be found in Chapter 10.

You should identify which type of license you need and then provide ETAS with the appropriate information as follows:

Machine-named licenses allows RTA-OS to be used by any user logged onto the PC on which RTA-OS and the machine-named license is installed.

A machine-named license can be issued by ETAS when you provide the host ID (Ethernet MAC address) of the host PC

User-named licenses allow the named user (or users) to use RTA-OS on any PC in the network domain.

A user-named license can be issued by ETAS when you provide the Windows user-name for your network domain.

Concurrent licenses allow any user on any PC up to a specified number of users to use RTA-OS. Concurrent licenses are sometimes called *floating* licenses because the license can *float* between users.

A concurrent license can be issued by ETAS when you provide the following information:

1. The name of the server
2. The Host ID (MAC address) of the server.
3. The TCP/IP port over which your FLEXnet license server will serve licenses. A default installation of the FLEXnet license server uses port 27000.



Figure 2.2: Obtaining License Information

You can use the ETAS License Manager to get the details that you must provide to ETAS when requesting a machine-named or user-named license and (optionally) store this information in a text file.

Open the ETAS License Manager and choose **Tools → Obtain License Info** from the menu. For machine-named licenses you can then select the network adaptor which provides the Host ID (MAC address) that you want to use as shown in Figure 2.2. For a user-based license, the ETAS License Manager automatically identifies the Windows username for the current user.

Selecting “Get License Info” tells you the Host ID and User information and lets you save this as a text file to a location of your choice.

2.3.3 Installing a Concurrent License Server

Concurrent licenses are allocated to client PCs by a FLEXnet license server manager working together with a vendor daemon. The vendor daemon for ETAS is called ETAS.exe. A copy of the vendor daemon is placed on disk when you install the ETAS License Manager and can be found in:

C:\Program Files\Common Files\ETAS\Licensing\Utility

To work with an ETAS concurrent license, a license server must be configured which is accessible from the PCs wishing to use a license. The server must be configured with the following software:

- FLEXnet license server manager;
- ETAS vendor daemon (ETAS.exe);

It is also necessary to install your concurrent license on the license server.

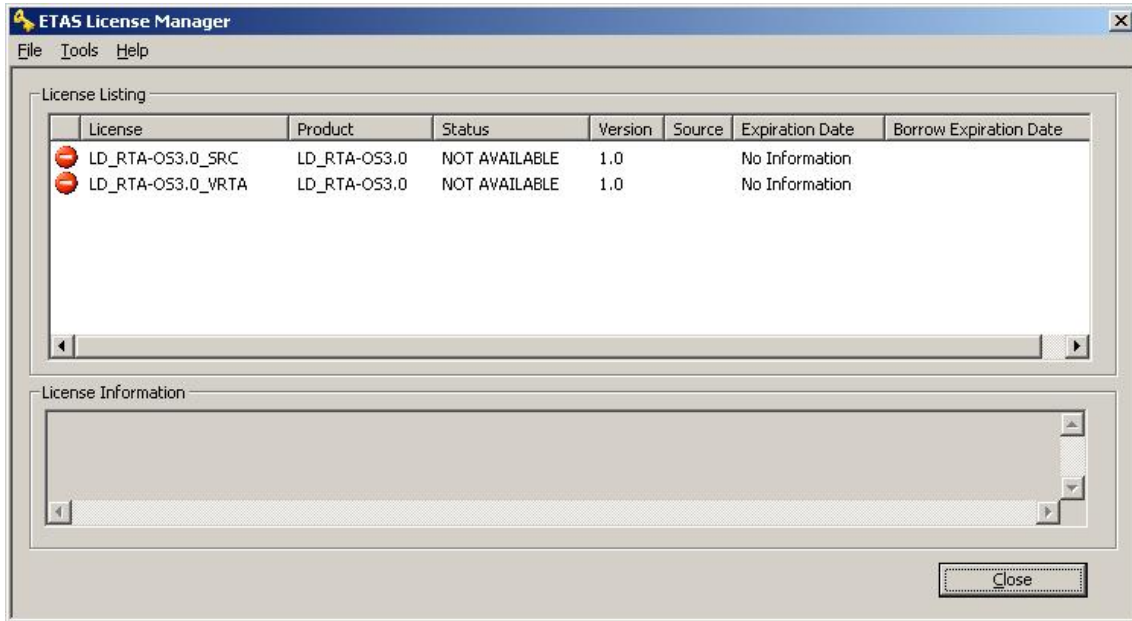


Figure 2.3: Unlicensed RTA-OS Installation

In most organizations there will be a single FLEXnet license server manager that is administered by your IT department. You will need to ask your IT department to install the ETAS vendor daemon and the associated concurrent license.

If you do not already have a FLEXnet license server then you will need to arrange for one to be installed. A copy of the FLEXnet license server, the ETAS vendor daemon and the instructions for installing and using the server (LicensingEndUserGuide.pdf) are placed on disk when you install the ETAS License manager and can be found in:

C:\Program Files\Common Files\ETAS\Licensing\Utility

2.3.4 Using the ETAS License Manager

If you try to run the RTA-OS GUI **rtaoscfg** without a valid license, you will be given the opportunity to start the ETAS License Manager and select a license. (The command-line tool **rtaosgen** will just report the license is not valid.)

When the ETAS License Manager is launched, it will display the RTA-OS license state as NOT AVAILABLE. This is shown in Figure 2.3.

Note that if the ETAS License Manager window is slow to start, **rtaoscfg** may ask a second time whether you want to launch it. You should ignore the request until the ETAS License Manager has opened and you have completed the configuration of the licenses. You should then say yes again, but you can then close the ETAS License Manager and continue working.

License Key Installation

License keys are supplied in an ASCII text file, which will be sent to you on completion of a valid license agreement.

If you have a machine-based or user-based license key then you can simply install the license by opening the ETAS License Manager and selecting **File → Add License File** menu.

If you have a concurrent license key then you will need to create a license stub file that tells the client PC to look for a license on the FLEXnet server as follows:

1. create a copy of the concurrent license file
2. open the copy of the concurrent license file and delete every line *except* the one starting with SERVER
3. add a new line containing USE_SERVER
4. add a blank line
5. save the file

The file you create should look something like this:

```
SERVER <server name> <MAC address> <TCP/IP Port>¶  
USE_SERVER¶  
¶
```

Once you have create the license stub file you can install the license by opening the ETAS License Manager and selecting **File → Add License File** menu and choosing the license stub file.

License Key Status

When a valid license has been installed, the ETAS License Manager will display the license version, status, expiration date and source as shown in Figure 2.4.

Borrowing a concurrent license

If you use a concurrent license and need to use RTA-OS on a PC that will be disconnected from the network (for example, you take a demonstration to a customer site), then the concurrent license will not be valid once you are disconnected.

To address this problem, the ETAS License Manager allows you to temporarily borrow a license from the license server.

To borrow a license:

1. Right click on the license feature you need to borrow.
2. Select "Borrow License"
3. From the calendar, choose the date that the borrowed license should expire.
4. Click "OK"

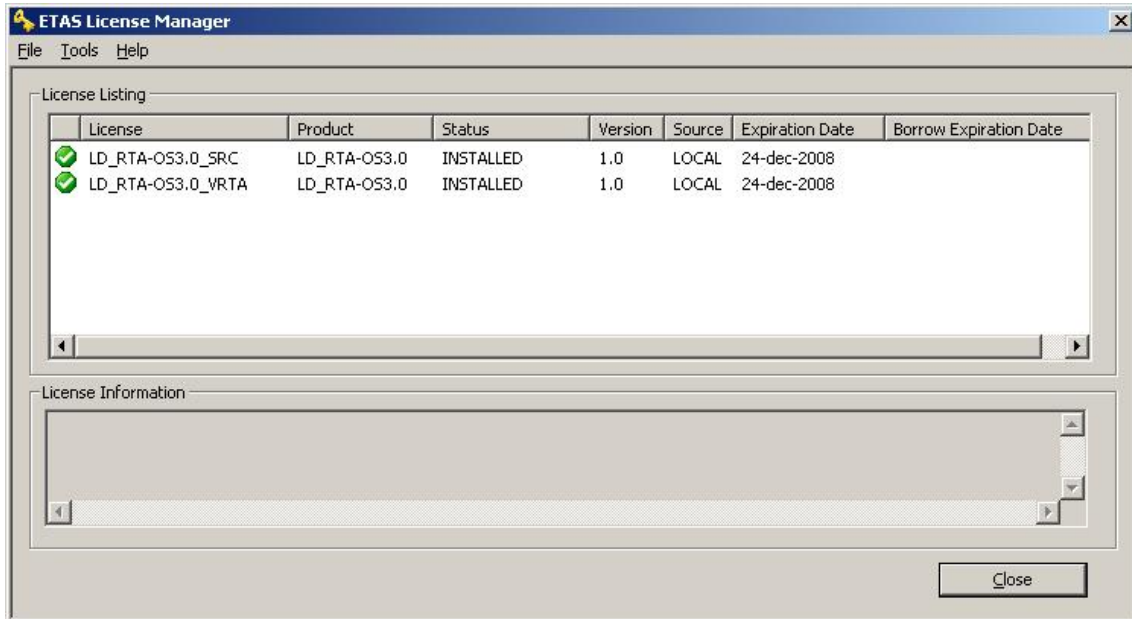


Figure 2.4: Licensed features for RTA-OS

The license will automatically expire when the borrow date elapses. A borrowed license can also be returned before this date. To return a license:

1. Reconnect to the network;
2. Right-click on the license feature you have borrowed;
3. Select "Return License".

2.3.5 Troubleshooting Licenses

RTA-OS tools will report an error if you try to use a feature for which a correct license key cannot be found. If you think that you should have a license for a feature but the RTA-OS tools appear not to work, then the following troubleshooting steps should be followed before contacting ETAS:

Can the ETAS License Manager see the license?

The ETAS License Manager must be able to see a valid license key for each product or product feature you are trying to use.

You can check what the ETAS License Manager can see by starting it from the **Help → License Manager...** menu option in **rtaoscfg** or directly from the Windows Start Menu - **Start → ETAS → License Management → ETAS License Manager**.

The ETAS License Manager lists all license features and their status. Valid licenses have status **INSTALLED**. Invalid licenses have status **NOT AVAILABLE**.

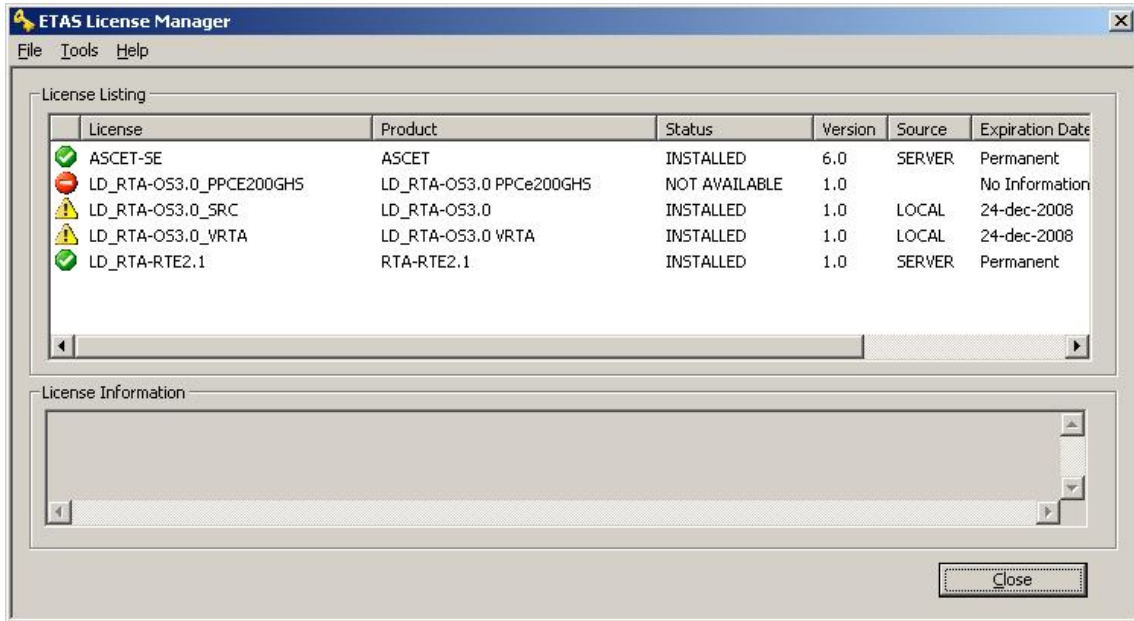


Figure 2.5: Licensed features that are due to expire

Is the license valid?

You may have been provided with a time-limited license (for example, for evaluation purposes) and the license may have expired. You can check that the Expiration Date for your licensed features to check that it has not elapsed using the ETAS License Manager.

If a license is due to expire within the next 30 days, the ETAS License Manager will use a warning triangle to indicate that you need to get a new license. Figure 2.5 shows that the license features LD_RTA-OS3.0_VRTA and LD_RTA-OS3.0_SRC are due to expire.

If your license has elapsed then please contact your local ETAS sales representative to discuss your options.

Does the Ethernet MAC address match the one specified?

If you have a machine based license then it is locked to a specific MAC address. You can find out the MAC address of your PC by using the ETAS License Manager (**Tools → Obtain License Info**) or using the Microsoft program **ipconfig /all** at a Windows Command Prompt.

You can check that the MAC address in your license file by opening your license file in a text editor and checking that the HOSTID matches the MAC address identified by the ETAS License Manager or the *Physical Address* reported by **ipconfig /all**.

If the HOSTID in the license file (or files) does not match your MAC address then you do not have a valid license for your PC. You should contact your local ETAS sales representative to discuss your options.

Is your Ethernet Controller enabled?

If you use a laptop and RTA-OS stops working when you disconnect from the network then you should check your hardware settings to ensure that your Ethernet controller is not turned off to save power when a network connection is not present. You can do this using Windows Control Panel. Select **System → Hardware → Device Manager** then select your Network Adapter. Right click to open **Properties** and check that the Ethernet controller is not configured for power saving in **Advanced** and/or **Power Management** settings.

Is the FlexNet License Server visible?

If your license is served by a FlexNet license server, then the ETAS License Manager will report the license as NOT AVAILABLE if the license server cannot be accessed.

You should contact your IT department to check that the server is working correctly.

Still not fixed?

If you have not resolved your issues, after confirming these points above, please contact ETAS technical support. The contact address is provided in Section [10.1](#). You must provide the contents and location of your license file and your Ethernet MAC address.

3 Verifying your Installation

Now that you have installed both the RTA-OS tools and a port plug-in and have obtained and installed a valid license key you can check that things are working.

3.1 Checking the Port

The first thing to check is that the RTA-OS tools can see the new port. You can do this in two ways:

1. use the **rtaosgen** tool

You can run the command **rtaosgen --target:?** to get a list of available targets, the versions of each target and the variants supported, for example:

```
RTA-OS Code Generator
Version p.q.r.s, Copyright © ETAS nnnn
Available targets:
  TriCoreHighTec_n.n.n [TC1797...]
  VRTA_n.n.n [MinGW,VS2005,VS2008,VS2010]
```

2. use the **rtaoscfg** tool

The second way to check that the port plug-in can be seen is by starting **rtaoscfg** and selecting **Help → Information...** drop down menu. This will show information about your complete RTA-OS installation and license checks that have been performed.



Integration Guidance 3.1: *If the target port plug-ins have been installed to a non-default location, then the `--target_include` argument must be used to specify the target location.*

If the tools can see the port then you can move on to the next stage – checking that you can build an RTA-OS library and use this in a real program that will run on your target hardware.

3.2 Running the Sample Applications

Each RTA-OS port is supplied with a set of sample applications that allow you to check that things are running correctly. To generate the sample applications:

1. Create a new *working* directory in which to build the sample applications.
2. Open a Windows command prompt in the new directory.
3. Execute the command:

```
rtaosgen --target:<your target> --samples:[Applications]
```

e.g.

```
rtaosgen --target:[MPC5777Mv2]PPCe200HighTec_5.0.8
--samples:[Applications]
```

You can then use the build.bat and run.bat files that get created for each sample application to build and run the sample. For example:

```
cd Samples\Applications\HelloWorld
build.bat
run.bat
```

Remember that your target toolchain must be accessible on the Windows PATH for the build to be able to run successfully.



Integration Guidance 3.2: *It is strongly recommended that you build and run at least the Hello World example in order to verify that RTA-OS can use your compiler toolchain to generate an OS kernel and that a simple application can run with that kernel.*

For further advice on building and running the sample applications, please consult your *Getting Started Guide*.

4 Port Characteristics

This chapter tells you about the characteristics of RTA-OS for the TraveoII/GHS port.

4.1 Parameters of Implementation

To be a valid OSEK (ISO 17356) or AUTOSAR OS, an implementation must support a minimum number of OS objects. The following table specifies the *minimum* numbers of each object required by the standards and the *maximum* number of each object supported by RTA-OS for the TraveoII/GHS port.

Parameter	Required	RTA-OS
Tasks	16	1024
Tasks not in SUSPENDED state	16	1024
Priorities	16	1024
Tasks per priority	-	1024
Queued activations per priority	-	4294967296
Events per task	8	32
Software Counters	8	4294967296
Hardware Counters	-	4294967296
Alarms	1	4294967296
Standard Resources	8	4294967296
Linked Resources	-	4294967296
Nested calls to GetResource()	-	4294967296
Internal Resources	2	no limit
Application Modes	1	4294967296
Schedule Tables	2	4294967296
Expiry Points per Schedule Table	-	4294967296
OS Applications	-	4294967295
Trusted functions	-	4294967295
Spinlocks (multicore)	-	4294967295
Register sets	-	4294967296

4.2 Configuration Parameters

Port-specific parameters are configured in the **General** → **Target** workspace of **rtaoscfg**, under the “Target-Specific” tab.

The following sections describe the port-specific configuration parameters for the TraveoII/GHS port, the name of the parameter as it will appear in the XML configuration and the range of permitted values (where appropriate).

4.2.1 Stack used for C-startup

XML name SpPreStartOS

Description

The amount of stack already in use at the point that StartOS() is called. This value is simply added to the total stack size that the OS needs to support all tasks and interrupts at run-time. Typically you use this to obtain the amount of stack that the linker must allocate. The value does not normally change if the OS configuration changes.

4.2.2 Stack used when idle

XML name SpStartOS

Description

The amount of stack used when the OS is in the idle state (typically inside Os_Cbk_Idle()). This is just the difference between the stack used at the point that Os_StartOS() is called and the stack used when no task or interrupt is running. This can be zero if Os_Cbk_Idle() is not used. It must include the stack used by any function called while in the idle state. The value does not normally change if the OS configuration changes.

4.2.3 Stack overheads for ISR activation

XML name SpIDisp

Description

The extra amount of stack needed to activate a task from within an ISR. If a task is activated within a Category 2 ISR, and that task has a higher priority than any currently running task, then for some targets the OS may need to use marginally more stack than if it activates a task that is of lower priority. This value accounts for that. On most targets this value is zero. This value is used in worst-case stack size calculations. The value may change if significant changes are made to the OS configuration. e.g. STANDARD/EXTENDED, SC1/2/3/4.

4.2.4 Stack overheads for ECC tasks

XML name SpECC

Description

The extra amount of stack needed to start an ECC task. ECC tasks need to save slightly more state on the stack when they are started than BCC tasks. This value contains the difference. The value may change if significant changes are made to the OS configuration. e.g. STANDARD/EXTENDED, SC1/2/3/4. Also note that if you are using stack repositioning (to align the stack of untrusted code to suit the MPU) then you will need to reduce the value by the amount of the adjustment.

4.2.5 Stack overheads for ISR

XML name SpPreemption

Description

The amount of stack used to service a Category 2 ISR. When a Category 2 ISR interrupts a task, it usually places some data on the stack. If the ISR measures the stack to determine if the preempted task has exceeded its stack budget, then it will overestimate the stack usage unless this value is subtracted from the measured size. The value is also used when calculating the worst-case stack usage of the system. Be careful to set this value accurately. If its value is too high then when the subtraction occurs, 32-bit underflow can occur and cause the OS to think that a budget overrun has been detected. The value may change if significant changes are made to the OS configuration. e.g. STANDARD/EXTENDED, SC1/2/3/4.

4.2.6 Orti22/Lauterbach

XML name Orti22Lauterbach

Description

Select ORTI 2.2 generation for the Lauterbach debugger.

Settings

Value	Description
true	Generate ORTI
false	No ORTI (default)

4.2.7 Orti23/Lauterbach

XML name Orti23Lauterbach

Description

Select ORTI 2.3 generation for the Lauterbach debugger.

Settings

Value	Description
true	Generate ORTI
false	No ORTI (default)

4.2.8 ORTI Stack Fill

XML name OrtiStackFill

Description

Expands ORTI information to cover stack address, size and fill pattern details to support debugger stack usage monitoring. Its use may depend on the debugger used.

Settings

Value	Description
true	Support ORTI stack tracking
false	ORTI stack tracking unsupported (default)

4.2.9 Block default interrupt

XML name block_default_interrupt

Description

Where a default interrupt is specified, it will normally execute if an unexpected interrupt fires. You can block this behavior using this option. The option affects the enabling of unused interrupt sources. If the default interrupt is not specified this option has no effect.

Settings

Value	Description
true	Block the default interrupt
false	Allow the default interrupt handler to run if an unexpected interrupt fires (default)

4.2.10 Enable stack repositioning

XML name AlignUntrustedStacks

Description

Use to support realignment of the stack for untrusted code when there are MPU protection region granularity issues. Refer to the documentation for Os_Cbk_SetMemoryAccess.

Settings

Value	Description
true	Support repositioning
false	Normal behavior (default)

4.2.11 GetAbortStack always

XML name always_call_GetAbortStack

Description

When the abort ISR is triggered always use the Os_Cbk_GetAbortStack() callback to set up a safe area of memory to use as a stack executing the ProtectionHook (please refer to the documentation for Os_Cbk_GetAbortStack).

Settings

Value	Description
true	Always call Os_Cbk_GetAbortStack()
false	Only call Os_Cbk_GetAbortStack() when the 'Enable untrusted stack check' target option is selected (default)

4.2.12 Alignment boundary setting

XML name alignment_boundary

Description

Applies either the Green Hills compiler option '-align4' or '-align8' (see the compiler documentation for more details).

Settings

Value	Description
4	Use -align4
8	Use -align8 (default)

4.2.13 Link Type

XML name OsLinkerModel

Description

Select the type of map used in linker samples. ('FLASH'=Code and data in FLASH. 'RAM'=Code and data in RAM (default). 'TCM'=Code and data in TCM RAM (Cortex-M7 cores only)).

Settings

Value	Description
FLASH	Code and data in FLASH
RAM	Code and data in RAM (default)

4.2.14 Set floating-point mode

XML name FloatingPointMode

Description

Select the floating-point mode used by the compiler (see the compiler documentation for more details).

Settings

Value	Description
fhard	Floating-point operations performed in hardware
fnone	Floating-point operations disallowed
fsingle	Single-precision floating-point operations performed in hardware the rest in software
fsoft	Floating-point operations performed in software (default)

4.2.15 Set floating-point support

XML name FloatingPointSupport

Description

Extra code can be added during context switches to preserve floating point registers used by tasks and ISRs. This option should only be used if the majority of tasks and ISRs in the application contain FPU instructions otherwise the register set mechanism may be more efficient. This mechanism does NOT use lazy saving of the FPU state and applies the GHS compiler option 'fhard' or 'fsingle' when needed. Refer to compiler documentation for more information.

Settings

Value	Description
none	The OS does not interact with the FPU (default)
autosave	Save floating-point registers S0 to S15 and the FPSCR at the start of an exception or interrupt

4.2.16 Stack protector setting

XML name stack_protector

Description

Enable protection against stack smashing attacks using the compiler -stack_protector command line option (see the compiler documentation for more details).

Settings

Value	Description
true	Compile with -stack_protector option
false	Compile without -stack_protector option (default)

4.2.17 Hide spurious interrupt

XML name hide_spurious_interrupt

Description

When enabled, spurious peripheral System Interrupt Controller (SIC) interrupts are ignored.

Settings

Value	Description
true	Spurious SIC interrupts are ignored
false	Spurious SIC interrupts are routed to either the spurious interrupt handler if defined, the default interrupt if present, or the unexpected interrupt handler (default)

4.2.18 Clear demux isrs when lowering ipl

XML name clear_demux_isrs_when_lowering_ipl

Description

When enabled, spurious peripheral SIC interrupts are cleared in the NVIC by clearing the pending flags of the Demux interrupts. This has a small performance overhead so is made optional. This can be used to prevent spurious interrupts occurring as a result of interrupts being cleared before they have been handled in application code.

Settings

Value	Description
true	Demux ISRs NVIC pending flags cleared when lowering the IPL
false	No additional clearing when lowering the IPL (default)

4.2.19 CrossCore0 IPC

XML name CrossCore0IPC

Description

Optionally specify the CPUSS Inter-Processor Communication (IPC) Interrupt used for cross-core interrupts for core CM_0. A free IPC interrupt will be selected automatically if one is not specified. Used in multicore applications only.

Settings

Value	Description
SIC0	Lowest System Interrupt Controller interrupt channel used for IPC interrupts
SIC7	Highest System Interrupt Controller interrupt interrupt channel used for IPC interrupts (all values between are valid)

4.2.20 CrossCore1 IPC

XML name CrossCore1IPC

Description

Optionally specify the CPUSS Inter-Processor Communication (IPC) Interrupt used for cross-core interrupts for core CM_1. A free IPC interrupt will be selected automatically if one is not specified. Used in multicore applications only.

Settings

Value	Description
SIC0	Lowest System Interrupt Controller interrupt channel used for IPC interrupts
SIC7	Highest System Interrupt Controller interrupt interrupt channel used for IPC interrupts (all values between are valid)

4.2.21 SpinLock IPC

XML name SpinLockIPC

Description

Optionally specify the CPUSS Inter-Processor Communication (IPC) channel used to support spinlocks between cores to protect data access. Used in multicore applications only.

Settings

Value	Description
IPC0	Lowest IPC Channel 0
IPC7	Highest IPC Channel 7 (default, all values between are valid)

4.2.22 Use Default VTOR

XML name use_default_VTOR

Description

The default VTOR value is contained in the CPUSS_CMx_VECTOR_TABLE_BASE register when the Cortex-M4F/M7 CPU is brought out of reset. If the default value is used it is the user’s responsibility to correctly place the first two entries of the vector table (i.e. the Initial SP Value and the Reset Vector) and the generated vector table immediately after those values. If the default VTOR value is not used then the vector table will use NULL_PTR’s for the first two entries and set the VTOR using the address of the generated vector table. The option affects the size of the generated vector table, and setting of VTOR in ‘Os_InitializeVectorTable’.

Settings

Value	Description
true	Omit the Initial SP Value and Reset Vector from the vector table, ‘Os_InitializeVectorTable’ does NOT set VTOR (default)
false	Initial SP Value and Reset Vector are part of the generated vector table, VTOR Register is set in ‘Os_InitializeVectorTable’

4.3 Generated Files

The following table lists the files that are generated by **rtaosgen** for all ports:

Filename	Contents
Os.h	The main include file for the OS.
Os_Cfg.h	Declarations of the objects you have configured. This is included by Os.h.
Os_MemMap.h	AUTOSAR memory mapping configuration used by RTA-OS to merge with the system-wide MemMap.h file in AUTOSAR versions 4.0 and earlier. From AUTOSAR version 4.1, Os_MemMap.h is used by the OS instead of MemMap.h.
RTAOS.<lib>	The RTA-OS library for your application. The extension <lib> depends on your target.
RTAOS.<lib>.sig	A signature file for the library for your application. This is used by rtaosgen to work out which parts of the kernel library need to be rebuilt if the configuration has changed. The extension <lib> depends on your target.
<projectname>.log	A log file that contains a copy of the text that the tool and compiler sent to the screen during the build process.

5 Port-Specific API

The following sections list the port-specific aspects of the RTA-OS programmers reference for the TraveoII/GHS port that are provided either as:

- additions to the material that is documented in the *Reference Guide*; or
- overrides for the material that is documented in the *Reference Guide*. When a definition is provided by both the *Reference Guide* and this document, the definition provided in this document takes precedence.

5.1 API Calls

5.1.1 Os_AwaitStartup

Routine used on a multicore variant by a non-master core to wait until the master core is ready for the core to start.

Syntax

```
FUNC(void, OS_APPL_CODE) Os_AwaitStartup(void)
```

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	all	The core does not exist or can not be started.

Description

This target port assumes that when the Cortex-M7 CPUs come out of reset all cores start at the same time. The `Os_AwaitStartup()` function is called by non-master cores after they have setup the stack and the small-data area registers to suspend execution until the master core calls `Os_Cbk_StartCore()` to start the core.

RTA-OS provides a default implementation of `Os_AwaitStartup()` that will be appropriate for most normal situations. In the RTA-OS samples, `Os_AwaitStartup()` is called from the code in `target_init.s` that runs before the `main()` function is called. It can also be called in `OS_MAIN()`.

Note that with the default implementations of `Os_AwaitStartup()` and `Os_Cbk_StartCore()` provided, your code is responsible for calling `Os_AwaitStartup()`.

If you provide your own version of `Os_AwaitStartup()` you will most likely need to provide your own version of `Os_Cbk_StartCore()` as well.

`Os_AwaitStartup()` should not get called for core 0 as this is the primary core.

Example

```
#define OS_START_CORE_KEY (0xAA551234U) /* Key that will not match a
    hardware self test pattern. */

OS_VOLATILE uint32 Os_CoresStarted[...];

FUNC(void, OS_APPL_CODE) Os_AwaitStartup(void) {
{
    CoreIdType core = (CoreIdType)((OS_CPUSS_IDENTITY >> 8) & 0x1U);

    /* Make the secondary core wait until activated by the master core. */
    while ((core == 1U) && (Os_CoresStarted[core] != OS_START_CORE_KEY)) {
        /* Issue a WFE instruction to put the secondary core into a low
         * power state. Note starting other cores, debug and semihosting
         * events can cause a SEV to wake-up the core. Check if it is
         * valid to wake-up this core. */
        __WFE();
    }

    /* Clear Os_CoresStarted[core] so that this function will have to wait
     * for Os_Cbk_StartCore(core) to be called again if the core is reset
     * and memory is not zeroed. */
    Os_CoresStarted[core] = 0U;

    __DSB();
}
}
```

Calling Environment

Tasks/ISRs	AUTOSAR OS Hooks	RTA-OS Hooks
Task	PreTaskHook	StackOverrunHook
Category 1 ISR	PostTaskHook	TimeOverrunHook
Category 2 ISR	StartupTaskHook	
	ShutdownHook	
	ErrorHook	
	ProtectionHook	

See Also

- StartCore
- StartNonAutosarCore
- StartOS
- Os_Cbk_StartCore

5.1.2 Os_InitializeVectorTable

Initialize the vector table.

Syntax

```
void Os_InitializeVectorTable(void)
```

Description

Os_InitializeVectorTable() initializes the NVIC and System interrupt controller according to the project configuration. This includes the CPU exception handlers and setting the STKALIGN and NONBASETHRDENA bits in the CCR, as these must be set for the OS to work as designed on this target. If RTA-OS has been configured to generate a vector table, it also sets the VTOR register on M4 or M7 cores to the appropriate address (on M7 cores this may need to be set before the core is started). It then enables Category 1 ISRs but leaves Category 2 ISRs disabled until after calling StartOS().

Os_InitializeVectorTable() should be called before StartOS(). It should be called even if 'Suppress Vector Table Generation' is set to TRUE. In multicore applications Os_InitializeVectorTable() should be called by all cores.

Example

```
Os_InitializeVectorTable();
```

See Also

StartOS

5.2 Callbacks

5.2.1 Os_Cbk_GetAbortStack

Callback routine to provide the start address of the stack to use for some exception conditions.

Syntax

```
FUNC(void *, OS_APPL_CODE) Os_Cbk_GetAbortStack(void)
```

Return Values

The call returns values of type **void ***.

Description

Untrusted code can misbehave and cause a protection exception. When this happens, AUTOSAR requires that ProtectionHook is called and the task, ISR or OS Application must be terminated.

It is possible that at the time of the fault the stack pointer is invalid. For this reason, if 'GetAbortStack always' is configured, RTA-OS will call Os_Cbk_GetAbortStack to get the address of a safe area of memory that it should use for the stack while it performs this processing.

If a value of zero is returned, the os does not adjust the stack.

The stack only needs to be large enough to perform the ProtectionHook. Maskable interrupts are disabled during this process.

A default implementation of `Os_Cbk_GetAbortStack` is supplied in the RTA-OS library that returns the address of an area of static memory.

Example

```
FUNC(void *,OS_APPL_CODE) Os_Cbk_GetAbortStack(void) {
    /* 64-bit alignment is needed for EABI. */
    static uint64 abortstack[32U];
    return &abortstack[32U];
}
```

Required when

The callback must be present if the 'GetAbortStack always' target option is enabled.

5.2.2 `Os_Cbk_GetSetProtection`

Callback routine used to control the activation of the memory protection system.

Syntax

```
FUNC(boolean, {memclass})Os_Cbk_GetSetProtection(
    boolean enable
)
```

Return Values

The call returns values of type `boolean`.

Description

This callback is used in configurations that have OS Applications where `TrustedApplicationWithProtection` is true. It must return the state of the memory protection hardware at the point it was called (TRUE if enabled, FALSE otherwise). It must then enable or disable memory protection based on the incoming 'enable' value. It is used to switch between Trusted and TrustedApplicationWithProtection modes.

The callback is required for this target platform. (Some platforms such as the TriCore can provide separate memory protection sets for untrusted, trusted and trusted-with-protection modes and in that case the callback is not used.)

Note: `memclass` is `OS_APPL_CODE` for AUTOSAR 3.x, `OS_OS_CBK_GETSETPROTECTION_CODE` for AUTOSAR 4.1 and `OS_CALLOUT_CODE` otherwise.

Example

```
FUNC(boolean {memclass}) Os_Cbk_GetSetProtection(boolean enable) {
    boolean initial = MPU.enabled;
    MPU.enabled = enable;
    return initial;
}
```

Required when

The callback must be provided when memory protection is selected and there are OS Applications where TrustedApplicationWithProtection is true.

See Also

- Os_Cbk_CheckMemoryAccess
- CallTrustedFunction
- CallAndProtectFunction
- Os_Cbk_SetMemoryAccess

5.2.3 Os_Cbk_SetMemoryAccess

Callback routine used to prepare the memory protection system for a switch from trusted to access-restricted code (untrusted or trusted-with-protection).

Syntax

```
FUNC(void, {memclass})Os_Cbk_SetMemoryAccess(
    Os_UntrustedContextRefType ApplicationContext
)
```

Parameters

Name	Type	Mode	Description
ApplicationContext	Os_UntrustedContextRefType	in	A reference to a type that describes the untrusted context.

Description

This callback is provided so that you have full control over the memory protection hardware on your device, and so that you can decide the degree of protection that you want to apply on a particular project. For example, you may choose to limit write-access for access-restricted code but allow any read and execute access. Alternatively you may wish to limit read/write and execute access for access-restricted code.

In an AUTOSAR OS, code that runs in the context of a Trusted OS Application is assumed to have full access to any area of RAM, ROM or IO space that is available. Such code runs in a privileged mode. On the other hand, code that runs in the context of an access-restricted OS Application may have restrictions placed on it that prevent it from being able access certain areas. Such code typically runs in 'user' (untrusted) mode, but there is also an AUTOSAR option to configure trusted OS Applications to run with memory protection enabled. Trusted-with-protection code behaves in most ways like trusted code. The only difference is that it runs with restricted access to memory.

Whenever RTA-OS is about to switch from trusted to access-restricted code, it makes a call to Os_Cbk_SetMemoryAccess. It passes in a reference to an

Os_UntrustedContextType data structure that you can use to determine what permissions to set for access-restricted code. The Os_UntrustedContextType structure contains information about the OS Application, Task/ISR and stack region that applies to the code that is about to be executed. Depending on the context of the switch, some of these may contain NULL values. Os_Cbk_SetMemoryAccess is only called from trusted code.

Os_Cbk_SetMemoryAccess gets called in the following cases:

- 1) Before calling a TASK that belongs to an access-restricted OS-Application.
- 2) Before calling a Category 2 ISR that belongs to an access-restricted OS-Application.
- 3) Before calling an access-restricted OS-Application Startup, Shutdown or Error hook.
- 4) Before calling a 'TrustedFunction' that belongs to an access-restricted OS-Application. (This extends the AUTOSAR concept, and allows a core trusted task to call out to access-restricted code supplied by third parties.)

When using memory protection features, you must initialize the memory protection hardware before calling StartOS(). You can choose what hardware to use, how many regions to protect and what restrictions to apply.

If you want to run all access-restricted code with the same memory protection settings, then you can set the 'Single Memory Protection Zone' OS option. In this case Os_Cbk_SetMemoryAccess will not be called. You must set up the MPU before running any access-restricted code.

If you want to run all access-restricted code with the same basic memory protection settings but apply protection to the stack, then you can set the 'Stack Only Memory Protection' OS option. In this case Os_Cbk_SetMemoryAccess will only be passed the stack-related fields (Address and Size) plus Application. You must ensure that the memory protection settings limit the stack to the specified range.

* Note * Where the hardware does not allow protection regions to be set at any address/size combination, you may choose to adjust the stack to a position that can be protected efficiently. For example, the protection region may have to be aligned on a 64-byte address boundary. In these cases, RTA-OS provides the 'AlignUntrustedStacks' configuration option. When this is set, a further field 'ApplicationContext->AlignedAddress' becomes available. Its initial value will be the same as ApplicationContext->Address. However you can change its value to signal to the OS that the access-restricted code should start at a different location. For the earlier example, if ApplicationContext->AlignedAddress initially has value 0x1020, you might change it to 0x1000 before returning so that the OS will start running the code at an address that is a multiple of 64. (This example assumes that the stack grows towards lower addresses.) You will have set the stack protection region to start from 0x1000.

* Note *

'FunctionID' and 'FunctionParams' are only present when there are access-restricted functions. The value of 'FunctionID' will be INVALID_FUNCTION except when the call-back is for an access-restricted function. In this case, 'FunctionID' contains the function identifier and 'FunctionParams' is a copy of the pointer to the parameters of the function.

* Note *

'CoreID' is only present where there are multiple AUTOSAR cores, and it holds the number of the current core.

Note: memclass is OS_APPL_CODE for AUTOSAR 3.x, OS_OS_CBK_SETMEMORYACCESS_CODE for AUTOSAR 4.1 and OS_CALLOUT_CODE otherwise.

Example

```

FUNC(void, {memclass}) Os_Cbk_SetMemoryAccess(Os_UntrustedContextRefType
ApplicationContext) {
/*
 * When called for an access-restricted TASK:
 * ApplicationContext->Application contains the ID of the OS
   Application that the TASK belongs to.
 * ApplicationContext->TaskID is the ID of the TASK
 * ApplicationContext->ISRID is INVALID_ISR
 * ApplicationContext->Address is the starting address for the TASK's
   stack.
 * ApplicationContext->Size is the stack budget configured for the
   TASK. (Zero if no budget.)
 * ApplicationContext->Trusted is true if the OS Application is trusted
   (TrustedApplicationWithProtection)
 *
 * When called for an access-restricted ISR:
 * ApplicationContext->Application contains the ID of the OS
   Application that the ISR belongs to.
 * ApplicationContext->TaskID is INVALID_TASK
 * ApplicationContext->ISRID is the ID of the ISR
 * ApplicationContext->Address is the starting address for the ISR's
   stack.
 * ApplicationContext->Size is the stack budget configured for the ISR.
   (Zero if no budget.)
 * ApplicationContext->Trusted is true if the OS Application is trusted
   (TrustedApplicationWithProtection)
 *
 * When called for:
 * - an access-restricted Function
 * - an access-restricted OS Application error hook
 * - an access-restricted OS Application startup hook
 * - an access-restricted OS Application shutdown hook
 * ApplicationContext->Application contains the ID of the OS
   Application that the function/hook belongs to.
 * ApplicationContext->TaskID is INVALID_TASK

```

```

* ApplicationContext->ISRID is INVALID_ISR
* ApplicationContext->Address is the value of the stack pointer just
  before the access-restricted code gets called.
* ApplicationContext->Size is zero
* ApplicationContext->Trusted is true if the OS Application is trusted
  (TrustedApplicationWithProtection)
*
* Where there are access-restricted Functions, there are two more
  fields:
* ApplicationContext->FunctionID contains the ID of the function
  (INVALID_FUNCTION unless being called for an access-restricted
  Function)
* ApplicationContext->FunctionParams contains FunctionParams for the
  access-restricted Function call (undefined for INVALID_FUNCTION)
*
* Be aware that on some target devices (Power PC, for example) the EABI
  might specify that a
* back link will be written before the stack pointer on entry.
* You will have to account for this in your calculations.
*
* For a multicore system, ApplicationContext->CoreID contains the ID of
  the calling core.
* This is omitted if the OS is only running on one core.
*/

/* Force AlignedAddress to the the next 64-byte value below Address */
(uint32)ApplicationContext->AlignedAddress &=
    ((uint32)ApplicationContext->Address % 64U);
SET_STACK_RANGE(ApplicationContext->AlignedAddress, STACK_ALLOWANCE);

if (ApplicationContext->Application == App2) {
    /* Set memory protection regions that apply for the overall
       application 'App2' */
    SET_UNTRUSTED_WRITE_RANGE(App2_BASE, App2_SIZE); /* Example */
    if (ApplicationContext->TaskID == App2TaskB) {
        /* Extend or restrict ranges as desired for Task 'App2TaskB' */
    }
    if (ApplicationContext->ISRID == App2ISR1) {
        /* Extend or restrict ranges as desired for ISR 'App2ISR1' */
    }
    if (ApplicationContext->FunctionID == UTF1) {
        /* Extend or restrict ranges as desired for access-restricted
           Function 'tf1' */
    }
}
if (ApplicationContext->Application == App3) {
    /* Set memory protection regions that apply for the overall
       application 'App3' */
    SET_UNTRUSTED_WRITE_RANGE(App3_BASE, App3_SIZE); /* Example */
    if (ApplicationContext->TaskID == App3TaskB) {
        /* Extend or restrict ranges as desired for Task 'App3TaskB' */
    }
}

```

```

    if (ApplicationContext->FunctionID == UTF2) {
        /* Extend or restrict ranges as desired for access-restricted
           Function 'tf2' */
    }
    if (ApplicationContext->FunctionID == UTF3) {
        /* Extend or restrict ranges as desired for access-restricted
           Function 'tf3' */
    }
}
...
}
OS_MAIN() {
    ...
    InitializeMemoryProtectionHardware();
    ...
    StartOS(OSDEFAULTAPPMODE);
}

```

Required when

The callback must be provided when memory protection is selected and there are access-restricted OS Applications.

See Also

- Os_Cbk_CheckMemoryAccess
- CallTrustedFunction
- CallAndProtectFunction

5.2.4 Os_Cbk_StartCore

Callback routine used to start a non-master core on a multi-core variant.

Syntax

```

FUNC(StatusType, {memclass})Os_Cbk_StartCore(
    uint16 CoreID
)

```

Return Values

The call returns values of type StatusType.

Value	Build	Description
E_OK	all	No error.
E_OS_ID	all	The core does not exist or can not be started.

Description

In a multi-core application, the StartCore() or StartNonAutosarCore() OS APIs have to be called prior to StartOS() for each core that is to run. For this target port, these APIs make a call to Os_Cbk_StartCore() which is responsible for starting the specified core.

RTA-OS provides a default implementation of Os_Cbk_StartCore() that will be appropriate for most normal situations. Os_Cbk_StartCore() does not get called for core 0, because core 0 must start first.

If you provide your own version of Os_Cbk_StartCore() you will most likely need to provide your own version of Os_AwaitStartup() as well.

Note: memclass is OS_APPL_CODE for AUTOSAR 3.x, OS_OS_CBK_STARTCORE_CODE for AUTOSAR 4.1 and OS_CALLOUT_CODE otherwise.

Example

```

FUNC(StatusType, {memclass}) Os_Cbk_StartCore(uint16 CoreID)
{
    StatusType ret = E_OS_ID;

    OS_VOLATILE uint32 *Os_CoreStartPtr = (OS_VOLATILE uint32
        *)&Os_StartCoreFlags;

    if (CoreID == 1U) {
        Os_CoresStarted[CoreID] = OS_START_CORE_KEY;
        __DSB();
        __SEV();
        ret = E_OK;
    }

    return ret;
}

```

Required when

Required for non-master cores that will be started.

See Also

StartCore
 StartNonAutosarCore
 StartOS
 Os_AwaitStartup

5.3 Macros

5.3.1 CAT1_ISR

Macro that should be used to create a Category 1 ISR entry function. This may also be used on user functions placed on exception vectors such as the hard/memory/usage/bus fault handlers and the NMI. (See the later section on "Writing Category 1

Interrupt Handlers" for more information). This macro exists to help make your code portable between targets.

Example

```
CAT1_ISR(MyISR) { ... }
```

5.3.2 Os_ClearDemuxInterrupts

Use of the `Os_ClearDemuxInterrupts` macro will clear the interrupt request bit of the NVIC Demux channels of the CPU. To use the `Os_ClearDemuxInterrupts` macro the file `Os_ConfigInterrupts.h` must be included through the use of `#include`. The macro is provided so that the Demux interrupt channels can be cleared when terminating an application. This prevents spurious interrupts as a result of pending SIC interrupts when the application is terminated. It may not be used by untrusted code.

Example

```
Os_ClearDemuxInterrupts()
```

5.3.3 Os_Clear_x

Use of the `Os_Clear_x` macro will clear the interrupt request bit of the named interrupt channel. The macro can be called using either the SIC channel number or the RTA-OS configured vector name. In the example, this is `Os_Clear_SIC48()` and `Os_Clear_Millisecond()` respectively. To use the `Os_Clear_x` macro the file `Os_ConfigInterrupts.h` must be included through the use of `#include`. The macro is provided so the interrupt channel can be cleared without corrupting the interrupt priority value configured by calling `Os_InitializeVectorTable()`. It may not be used by untrusted code.

Example

```
Os_Clear_SIC48()
Os_Clear_Millisecond()
```

5.3.4 Os_DisableAllConfiguredInterrupts

The `Os_DisableAllConfiguredInterrupts` macro will disable all configured SIC interrupt channels. To use the `Os_DisableAllConfiguredInterrupts` macro the file `Os_ConfigInterrupts.h` must be included through the use of `#include`. The macro is provided so the interrupt channels can be disabled without corrupting the interrupt priority values configured by calling `Os_InitializeVectorTable()`. It may not be used by untrusted code. In multi-core applications a macro is provided for each core (e.g. `Os_DisableAllConfiguredInterrupts_CPU0`).

Example

```
Os_DisableAllConfiguredInterrupts()
...
Os_EnableAllConfiguredInterrupts()
```

5.3.5 Os_Disable_x

Use of the `Os_Disable_x` macro will disable the named interrupt channel. The macro can be called using either the SIC channel number or the RTA-OS configured vector name. In the example, this is `Os_Disable_SIC48()` and `Os_Disable_Millisecond()` respectively. To use the `Os_Disable_x` macro the file `Os_ConfigInterrupts.h` must be included through the use of `#include`. The macro is provided so the interrupt channel can be masked without corrupting the interrupt priority value configured by calling `Os_InitializeVectorTable()`. It may not be used by untrusted code.

Example

```
Os_Disable_SIC48()
Os_Disable_Millisecond()
```

5.3.6 Os_EnableAllConfiguredInterrupts

The `Os_EnableAllConfiguredInterrupts` macro will enable all configured SIC interrupt channels. To use the `Os_EnableAllConfiguredInterrupts` macro the file `Os_ConfigInterrupts.h` must be included through the use of `#include`. The macro is provided so the interrupt channels can be enabled without corrupting the interrupt priority values configured by calling `Os_InitializeVectorTable()`. It may not be used by untrusted code. In multi-core applications a macro is provided for each core (e.g. `Os_EnableAllConfiguredInterrupts_CPU0`).

Example

```
Os_DisableAllConfiguredInterrupts()
...
Os_EnableAllConfiguredInterrupts()
```

5.3.7 Os_Enable_x

Use of the `Os_Enable_x` macro will enable the named interrupt channel. The macro can be called using either the SIC channel number or the RTA-OS configured vector name. In the example, this is `Os_Enable_SIC48()` and `Os_Enable_Millisecond()` respectively. To use the `Os_Enable_x` macro the file `Os_ConfigInterrupts.h` must be included through the use of `#include`. The macro is provided so the interrupt channel can be enabled without corrupting the interrupt priority value configured by calling `Os_InitializeVectorTable()`. It may not be used by untrusted code.

Example

```
Os_Enable_SIC48()
Os_Enable_Millisecond()
```

 5.3.8 Os_IntChannel_x

The `Os_IntChannel_x` macro can be used to get the vector number associated with the named SIC interrupt (0, 1, 2...). The macro can be called using either the SIC vector name or the RTA-OS configured vector name. In the example, this is `Os_IntChannel_TPU` and `Os_IntChannel_Millisecond` respectively. To use the `Os_IntChannel_x` macro the file `Os_ConfigInterrupts.h` must be included through the use of `#include`.

Example

```
trigger_interrupt(Os_IntChannel_TPU);
trigger_interrupt(Os_IntChannel_Millisecond);
```

 5.4 Type Definitions

 5.4.1 Os_StackSizeType

An unsigned value representing an amount of stack in bytes.

Example

```
Os_StackSizeType stack_size;
stack_size = Os_GetStackSize(start_position, end_position);
```

 5.4.2 Os_StackValueType

An unsigned value representing the position of the stack pointer (main stack).

Example

```
Os_StackValueType start_position;
start_position = Os_GetStackValue();
```

6 Toolchain

This chapter contains important details about RTA-OS and the GHS toolchain. A port of RTA-OS is specific to both the target hardware and a specific version of the compiler toolchain. You must make sure that you build your application with the supported toolchain.

In addition to the version of the toolchain, RTA-OS may use specific tool options (switches). The options are divided into three classes:

kernel options are those used by **rtaosgen** to build the RTA-OS kernel.

mandatory options must be used to build application code so that it will work with the RTA-OS kernel.

forbidden options must not be used to build application code.

Any options that are not explicitly forbidden can be used by application code providing that they do not conflict with the kernel and mandatory options for RTA-OS.

Integration Guidance 6.1: *ETAS has developed and tested RTA-OS using the tool versions and options indicated in the following sections. Correct operation of RTA-OS is only covered by the warranty in the terms and conditions of your deployment license agreement when using identical versions and options. If you choose to use a different version of the toolchain or an alternative set of options then it is your responsibility to check that the system works correctly. If you require a statement that RTA-OS works correctly with your chosen tool version and options then please contact ETAS to discuss validation possibilities.*



RTA-OS supports the Green Hills Software v2017.1.4 and v2019.1.4 compilation tools.

6.1 Compiler Versions

This port of RTA-OS has been developed to work with the following compiler(s):

6.1.1 Green Hills Software, Compiler v2017.1.4

Ensure that cctthumb.exe is on the path and that the appropriate environment variables have been set.

Tested on Green Hills Software, Compiler v2017.1.4

See also http://www.ghs.com/products/arm_development.html

6.1.2 Green Hills Software, Compiler v2019.1.4

Ensure that cctthumb.exe is on the path and that the appropriate environment variables have been set.

Tested on Green Hills Software, Compiler v2019.1.4

See also http://www.ghs.com/products/arm_development.html

If you require support for a compiler version not listed above, please contact ETAS.

6.2 Options used to generate this guide

6.2.1 Compiler

Name cctthumb.exe

Version v2017.1.4 Release Date Sun Apr 09 17:00:05 PDT 2017

Options

Kernel Options

The following options were used to build the RTA-OS kernel for the configuration that was used to generate the performance figures in this document. If you select different target options, then the values used to build the kernel might change. You can run a Configuration Summary report to check the values used for your configuration.

- cpu=cortexm7** Select the CPU type (may vary with CPU variant)
- float** Floating-point mode (value set by target option)
- C99** Use strict ISO/IEC 9899:1999 compliant C code
- align8** Align the stack to a 8-byte boundary (can be changed by a target option)
- asm3g** Enable use of `'.inspect'`, `'.struct'`, `'offsetof()'` and `'sizeof()'` in assembly language code
- delete** Remove functions that are unused and unreferenced
- gsize** Enable the `'gsize'` utility
- misalign_pack** Misaligned data access is supported
- Osize** Optimize for size
- preprocess_assembly_files** Run the C preprocessor on all assembly language files
- no_stack_protector** Do not enable compiler protection against stack smashing attacks (can be changed by a target option)
- Wimplicit-int** Enable warnings if function return type is not declared before use
- Wshadow** Enable warnings about local variables that have the same name as a global variable
- Wundef** Enable warnings about the use of undefined symbols in C preprocessor expressions
- incorrect_pragma_warnings** Enable warnings about `'#pragma'` directives that use the wrong syntax
- no_alternative_tokens** Disable use of alternative punctuation symbols
- no_commons** Assign uninitialized global variables to aligned sections

- **--no_coverage_analysis** Disable legacy coverage profiling
- **--no_wrap_diagnostics** Do not wrap diagnostic messages
- **--prototype_errors** Enable errors about functions that are referenced or called when no prototype is provided
- **--short_enum** Use smallest type possible for enumerated types
- **--unknown_pragma_warnings** Enable warnings about '#pragma' directives that are not recognized by the compiler
- **-check=none** Disable all run-time checks

Mandatory Options for Application Code

The following options were mandatory for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the values required by application code might change. You can run a Configuration Summary report to check the values used for your configuration.

- The same options as for kernel compilation with the exception of the following which may be omitted from application code:
 - brief_diagnostics,
 - no_wrap_diagnostics,
 - quit_after_warnings.

Forbidden Options for Application Code

The following options were forbidden for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the forbidden values might change. You can run a Configuration Summary report to check the values used for your configuration.

- **-no_short_enum** Store enumerations as integers
- Any other options that conflict with kernel options

6.2.2 Assembler

Name cctthumb.exe
Version v2017.1.4 Release Date Sun Apr 09 17:00:05 PDT 2017

Options

Kernel Options

The following options were used to build the RTA-OS kernel for the configuration that was used to generate the performance figures in this document. If you select different target options, then the values used to build the kernel might change. You can run a Configuration Summary report to check the values used for your configuration.

- The same options as for compilation

Mandatory Options for Application Code

The following options were mandatory for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the values required by application code might change. You can run a Configuration Summary report to check the values used for your configuration.

- The same options as for compilation

Forbidden Options for Application Code

The following options were forbidden for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the forbidden values might change. You can run a Configuration Summary report to check the values used for your configuration.

- Any options that conflict with kernel options

6.2.3 Librarian

Name ax.exe
Version v2017.1.4 Release Date Sun Apr 09 17:00:05 PDT 2017

6.2.4 Linker

Name elxr.exe
Version v2017.1.4 Release Date Sun Apr 09 17:00:05 PDT 2017

Options

Kernel Options

The following options were used to build the RTA-OS kernel for the configuration that was used to generate the performance figures in this document. If you select different target options, then the values used to build the kernel might change. You can run a Configuration Summary report to check the values used for your configuration.

- Man** Sort map file alphabetically and numerically.

- ML** Output locals to map file
- Mu** Output undefined symbols to the map file
- Mx** Output cross reference information to the map file
- keepmap** Keep the map file if linking fails
- mapfile_type=2** Output detailed information to the map file
- globalcheck=normal** Check definition and use of global variables
- globalcheck_qualifiers** Check const and volatile qualifiers of global variables
- strict_overlap_check** It is an error if sections overlap
- Qn** Do not print linker ID string
- e=reset_handler** Specify the application entry point

Mandatory Options for Application Code

The following options were mandatory for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the values required by application code might change. You can run a Configuration Summary report to check the values used for your configuration.

- The same options as for compilation

Forbidden Options for Application Code

The following options were forbidden for application code used with the configuration that was used to generate the performance figures in this document. If you select different target options, then the forbidden values might change. You can run a Configuration Summary report to check the values used for your configuration.

- Any options that conflict with kernel options

6.2.5 Debugger

Name Lauterbach TRACE32
Version Build 90814 or later

Notes on using ORTI with the Lauterbach debugger

When ORTI information for the Trace32 debugger is enabled entry and exit times for Category 1 interrupts are increased by a few cycles to support tracking of Category 1 interrupts by the debugger.

ORTI Stack Fill with the Lauterbach debugger

The 'ORTI Stack Fill' target option is provided to extend the ORTI support to allow evaluation of unused stack space. The Task.Stack.View command can then be used in the Trace32 debugger. The following must also be added to an application to ensure correct operation (as demonstrated in the sample applications):

The linker file must create labels holding the start address and stack size for each stack (one per core). The labels automatically generated by the linker can be used. For a single core system (i.e. core 0 only) the labels are:

The fill pattern used by the debugger must be contained within a 32 bit constant OS_STACK_FILL (i.e. for a fill pattern 0xCAFEF00D).

```
const uint32 OS_STACK_FILL = 0xCAFEF00D;
```

The stack must also be initialized with this fill pattern either in the application start-up routines or during debugger initialization.

Note that this feature may not be supported on all debugger versions.

7 Hardware

7.1 Supported Devices

This port of RTA-OS has been developed to work with the following target:

Name: Cypress Semiconductor
Device: TraveoII

The following variants of the TraveoII are supported:

- CYT2B6 (Covers the CYT2B6xxxx family running on the Cortex-M4F CPU)
- CYT2B7 (Covers the CYT2B7xxxx family running on the Cortex-M4F CPU)
- CYT2B9 (Covers the CYT2B9xxxx family running on the Cortex-M4F CPU)
- CYT2BL (Covers the CYT2BLxxxx family running on the Cortex-M4F CPU)
- CYT3BB (Covers the CYT3BBxxxx family)
- CYT3DL (Covers the CYT3DLxxxx family running on the Cortex-M4F CPU)
- CYT4BB (Multi-core CYT4BBxxxx variant running on the Cortex-M7 CM7_0 and CM7_1 CPUs)
- CYT4BB_CM7_0 (Single-core CYT4BBxxxx variant running on the Cortex-M7 CM7_0 CPU)
- CYT4BB_CM7_1 (Single-core CYT4BBxxxx variant running on the Cortex-M7 CM7_1 CPU)
- CYT4BF (Multicore CYT4BFxxxx variant running on the Cortex-M7 CM7_0 and CM7_1 CPUs)
- CYT4BF_CM7_0 (Single-core CYT4BFxxxx variant running on the Cortex-M7 CM7_0 CPU)
- CYT4BF_CM7_1 (Single-core CYT4BFxxxx variant running on the Cortex-M7 CM7_1 CPU)
- CYT4DN (Multicore CYT4DNxxxx variant running on the Cortex-M7 CM7_0 and CM7_1 CPUs)
- CYT4DN_CM7_0 (Single-core CYT4DNxxxx variant running on the Cortex-M7 CM7_0 CPU)
- CYT4DN_CM7_1 (Single-core CYT4DNxxxx variant running on the Cortex-M7 CM7_1 CPU)
- CYT4EN (Multicore CYT4ENxxxx variant running on the Cortex-M7 CM7_0 and CM7_1 CPUs)

- CYT4EN_CM7_0 (Single-core CYT4ENxxxx variant running on the Cortex-M7 CM7_0 CPU)
- CYT4EN_CM7_1 (Single-core CYT4ENxxxx variant running on the Cortex-M7 CM7_1 CPU)
- Generic_TVIM4F (Untested general Cortex-M4F variant with 1023 interrupt channels)
- Generic_TVIM7 (Untested general Cortex-M7 variant with 1023 interrupt channels)

If you require support for a variant of TraveoII not listed above, please contact ETAS.

7.2 Register Usage

7.2.1 Initialization

RTA-OS requires the following registers to be initialized to the indicated values before StartOS() is called.

Register	Setting
CCR.NONBASETHRDENA	This must be one so that the processor can enter Thread mode with exceptions active. Calling <code>Os_InitializeVectorTable()</code> will do this for you.
CCR.STKALIGN	This must be one so that 8-byte stack alignment is forced for exceptions. Calling <code>Os_InitializeVectorTable()</code> will do this for you.
CONTROL.FPCA	If the 'FloatingPointSupport' target option is set to 'autosave' then the FPCA bit must be set before calling <code>StartOS()</code> .
CONTROL.SPSEL	This must be cleared before calling <code>StartOS()</code> so that <code>SP_main</code> is used in thread mode.
CONTROL.nPRIV	This must be cleared before calling <code>StartOS()</code> so that the OS is entered in privileged mode.
CPACR	If the 'FloatingPointSupport' target option is set to 'autosave' then both the CP10 and CP11 fields in the CPACR must be set to allow full access.
FAULTMASK	This must be cleared before <code>StartOS()</code> . Calling <code>Os_InitializeVectorTable()</code> will do this for you.
FPCCR	If the 'FloatingPointSupport' target option is set to 'autosave' then the ASPEN bit of the FPCCR must be set and the LSPEN bit must be clear.
IPSR	This must be cleared before <code>StartOS()</code> or <code>Os_InitializeVectorTable()</code> .
NVIC	This must be initialized before <code>StartOS()</code> . Calling <code>Os_InitializeVectorTable()</code> will do this for you.
PRIMASK	This must be cleared before <code>StartOS()</code> . Calling <code>Os_InitializeVectorTable()</code> will do this for you.
SHPRx	These must be initialized before <code>StartOS()</code> . Calling <code>Os_InitializeVectorTable()</code> will do this for you.
SP (R13)	The stack must be allocated and SP initialized before calling <code>StartOS()</code> .
VTOR	If the 'Suppress Vector Table Generation' option is specified, the user must initialize the VTOR register before calling <code>StartOS()</code> . Otherwise, this will be set by calling <code>Os_InitializeVectorTable()</code> in Cortex-M4 cores. For Cortex-M7 cores VTOR may be set before the core is brought out of reset.

7.2.2 Modification

The following registers must not be modified by user code after the call to `StartOS()`:

Register	Notes
ACTLR	User code may not change the values in the ACTLR register.
BASEPRI	User code may not change the special-purpose BASEPRI register.
CCR	User code may not change the STKALIGN or NON-BASETHRDENA bits of the CCR.
CONTROL	User code may not change the values in the CONTROL register.
CPACR	If you intend to use hardware floating point instructions then both the CP10 and CP11 fields in the CPACR must be set to allow full access to the co-processors.
CPUSS_CMx_SYSTEM_INT_CTL	User code may not change the CPU_INT_IDX or ENABLED fields of the CPUSS_CMx_SYSTEM_INT_CTL registers
FAULTMASK	User code may not change the special-purpose FAULTMASK register.
FPCCR	User code may not make direct changes to FPCCR.
NVIC	User code may not make changes to the interrupt controller (NVIC) other than via OS APIs.
PRIMASK	User code may not change the special-purpose PRIMASK register.
SHCSR	User code may not make changes to SHCSR.
SHPRx	User code may not make changes to SHPRx.
SP (R13)	User code may not change the main stack pointer other than as a result of normal program flow.

7.3 Required OS resources

RTA-OS needs the following resources for correct operation.

Resource	Description
BusFault	The OS uses this to detect memory faults
HardFault	The OS uses this to detect protection faults
MemManage	The OS uses this to detect memory faults
SVCall	If untrusted code is present, the OS will use SVCall to move from unprivileged to privileged mode. It is also used when handling exceptions. If overridden by user-code, its behavior must be replicated.
UsageFault	The OS uses this to detect protection faults

RTA-OS runs Category 2 interrupts in thread mode. Because of this, RTA-OS cannot guarantee the state of the IPSR while a Category 2 interrupt is running. Software should make no assumptions about the state of the IPSR.

7.4 Interrupts

This section explains the implementation of RTA-OS's interrupt model on the TraveoII.

7.4.1 Interrupt Priority Levels

Interrupts execute at an interrupt priority level (IPL). RTA-OS standardizes IPLs across all targets. IPL 0 indicates task level. IPL 1 and higher indicate an interrupt priority. It is important that you don't confuse IPLs with task priorities. An IPL of 1 is higher than the highest task priority used in your application.

The IPL is a target-independent description of the interrupt priority on your target hardware. The following table shows how IPLs are mapped onto the hardware interrupt priorities of the TraveoII:

IPL	BASEPRI	Description
0	0x00	User (task) level. No interrupts are masked
1	0xE0	Maskable Category 1 and 2 interrupts routed through IRQ
2	0xC0	Maskable Category 1 and 2 interrupts routed through IRQ
.	Maskable Category 1 and 2 interrupts routed through IRQ
6	0x40	Maskable Category 1 and 2 interrupts routed through IRQ
7	0x20	Cortex-M CPU Category 1 MemoryManagement Fault, Bus Fault, and Usage Fault exceptions
8	n/a	Cortex-M CPU Category 1 SVCcall exception
9	n/a	Cortex-M CPU Category 1 HardFault exception
10	n/a	Cortex-M CPU Category 1 NMI exception

Even though a particular mapping is permitted, all Category 1 ISRs must have equal or higher IPL than all of your Category 2 ISRs.

7.4.2 Allocation of ISRs to Interrupt Vectors

The following restrictions apply for the allocation of Category 1 and Category 2 interrupt service routines (ISRs) to interrupt vectors on the TraveoII. A ✓ indicates that the mapping is permitted and a ✗ indicates that it is not permitted:

Address	Category 1	Category 2
CPU Trap2 exception handler (NMI handler)	✓	✗
CPU Trap3 exception handler (Hard Fault)	✓	✗
CPU Trap4 exception handler (Memory Management Fault)	✓	✗
CPU Trap5 exception handler (Bus Fault)	✓	✗
CPU Trap6 exception handler (Usage Fault)	✓	✗
CPU Trap11 exception handler (SVCcall)	✓	✗
CPU Trap12 exception handler (DebugMonitor)	✓	✗
CPU Trap14 exception handler (PendSV)	✓	✗
CPU Trap15 exception handler (SysTick)	✓	✓
NVIC interrupt handlers	✓	✓
SIC interrupt handlers	✓	✓

To facilitate further control over the interrupts Enable, Disable and Clear macros are provided for each configured NVIC, SIC and the SysTick interrupt.

7.4.3 Vector Table

rtaosgen normally generates an interrupt vector table for you automatically. You can configure "Suppress Vector Table Generation" as `true` to stop RTA-OS from generating the interrupt vector table.

Depending upon your target, you may be responsible for locating the generated vector table at the correct base address. The following table shows the section (or sections) that need to be located and the associated valid base address:

Section	Valid Addresses
<code>.os_exception_vectors</code>	Section containing exception vector table. Its first entry is the NMI vector (unless "Use Default VTOR" is used and set to false). It should be located 8 bytes above the start of the vector table. If "Use Default VTOR" is used and set to false then the first entry is the reset vector and it should be located to the start of the vector table. The alignment depends upon the number of vectors and should have a minimum of 128 byte alignment when there are up to 32 vectors. See the ARM documentation for further details.
<code>.os_exception_handlers</code>	Section containing exception handling code.

The RTA-OS generated vector table does not include the reset vector or the stack pointer. These should be added for an application and located immediately before the vector table. The vector table itself should be aligned to an 0x80 byte boundary (i.e. with the lowest 7 bits of its address zero), as required by the VTOR register.

When the default interrupt is configured the RTA-OS generated vector table contains entries for all supported interrupts for the selected chip variant. If the default interrupt is not configured then entries are created up the highest configured interrupt.

7.4.4 Writing Category 1 Interrupt Handlers

Raw Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves. RTA-OS provides an optional helper macro `CAT1_ISR` that can be used to make code more portable. Depending on the target, this may cause the selection of an appropriate interrupt control directive to indicate to the compiler that a function requires additional code to save and restore the interrupt context.

A Category 1 ISR therefore has the same structure as a Category 2 ISR, as shown below.

```
CAT1_ISR(Category1Handler) {
    /* Handler routine */
}
```

RTA-OS uses the `SVC` handler. While RTA-OS automatically optimizes the code included in the `SVC` handler based on the project configuration, the `SVC` handler is always used, and so this functionality must be supported if a user-provided `SVC` handler is used in any application.

7.4.5 Writing Category 2 Interrupt Handlers

Category 2 ISRs are provided with a C function context by RTA-OS, since the RTA-OS kernel handles the interrupt context itself. The handlers are written using the `ISR()` macro as shown below:

```
#include <Os.h>
ISR(MyISR) {
    /* Handler routine */
}
```

You must not insert a return from interrupt instruction in such a function. The return is handled automatically by RTA-OS.

7.4.6 Category 2 Software Interrupts

When handling Category 2 interrupts RTA-OS switches between handler and thread mode. The mode change prevents software interrupts re-triggering (i.e. when the interrupt re-triggers by directly writing to the NVIC registers within the interrupt handler the interrupt will not re-enter when the interrupt returns). This restriction only applies to a software interrupt re-triggering itself, all other interrupts are handled as expected.

7.4.7 Default Interrupt

The 'default interrupt' is intended to be used to catch all unexpected interrupts. All unused interrupts have their interrupt vectors directed to the named routine that you specify. The routine you provide is not handled by RTA-OS and must correctly handle the interrupt context itself. The handler must use the `CAT1_ISR` macro in the same way as a Category 1 ISR (see Section 7.4.4 for further details).

The TraveoII/GHS port does not enable unused interrupt channels when the default interrupt is used. `Os_InitializeVectorTable()` masks all unused NVIC/SIC interrupts. If you intend to trigger a default interrupt then the modification to the NVIC/SIC registers must occur after calling `Os_InitializeVectorTable()`.

7.5 Memory Model

The following memory models are supported:

Model	Description
Standard	The standard 32-bit EABI memory model is used.

7.6 Processor Modes

RTA-OS can run in the following processor modes:

Mode	Notes
Trusted	All trusted code runs in privileged mode (i.e. in handler mode, or in thread mode with <code>CONTROL.nPRIV</code> unset).
Untrusted	All untrusted code runs in unprivileged mode (i.e. in thread mode with <code>CONTROL.nPRIV</code> set).

7.7 Stack Handling

RTA-OS uses a single stack for all tasks and ISRs.

RTA-OS manages the main stack via the stack pointer. The process stack is never used by RTA-OS, and must not be used by user code.

7.8 Processor state when calling StartOS()

At StartOS() the following conditions should be true:

- The CPU must be operating in privileged mode (i.e. CONTROL.nPRIV unset).
- The CPU must be using the main stack (i.e. CONTROL.SPSEL unset).
- PRIMASK must be clear.
- FAULTMASK must be clear.
- BusFault, UsageFault and MemFault must be enabled in the SHCSR.
- The CCR.STKALIGN bit must be set to guarantee 8-byte SP alignment of exception entries.
- The Floating point configuration must match that of the application configuration (i.e. FPCCR and CONTROL.FPCA must match the target option selection).

These conditions can be applied by calling `Os_InitializeVectorTable()`. If any of these conditions are not met then `ShutdownOS()` will be called from `StartOS()`.

7.9 Spurious interrupts when termination an application

Due to the interaction of the ARM NVIC and the TraveoII System Interrupt Controller (SIC) spurious interrupts may be generated when an OS Application is terminated. To prevent spurious interrupts occurring there should be no pending SIC interrupts (i.e. all interrupt pending flags should be cleared) when an OS Application is terminated. To help prevent spurious interrupts the macro `Os_ClearDemuxInterrupts` can be used in the interrupt termination callback functions. To catch any spurious interrupts a special ISR can be configured in RTA-OS.

7.10 Use of the Inter-Processor Communication hardware in multicore applications

In multicore applications IPC channel 7 is reserved by default by RTAOS to instrument spinlocks between cores. This may be updated using the 'Spinlock IPC' target option.

In multicore applications the first two unused IPC interrupts are reserved by default by RTAOS for signaling between cores. This may be updated using the 'CrossCore0 IPC' and 'CrossCore1 IPC' target options. If there are insufficient free IPC interrupts then an error is generated.

8 Performance

This chapter provides detailed information on the functionality, performance and memory demands of the RTA-OS kernel. RTA-OS is highly scalable. As a result, different figures will be obtained when your application uses different sets of features. The figures presented in this chapter are representative for the TraveoII/GHS port based on the following configuration:

- There are 32 tasks in the system
- Standard build is used
- Stack monitoring is disabled
- Time monitoring is disabled
- There are no calls to any hooks
- Tasks have unique priorities
- Tasks are not queued (i.e. tasks are BCC1 or ECC1)
- All tasks terminate/wait in their entry function
- Tasks and ISRs do not save any auxiliary registers (for example, floating point registers)
- Resources are shared by tasks only
- The generation of the resource RES_SCHEDULER is disabled

8.1 Measurement Environment

The following hardware environment was used to take the measurements in this chapter:

Device	CYT4BB_CM7_1 on Cypress CYTVII-B-H-8M-176-CPU
CPU Clock Speed	250.0MHz
Stopwatch Speed	250.0MHz

8.2 RAM and ROM Usage for OS Objects

Each OS object requires some ROM and/or RAM. The OS objects are generated by **rtaosgen** and placed in the RTA-OS library. In the main:

- 0s_Cfg_Counters includes data for counters, alarms and schedule tables.
- 0s_Cfg contains the data for most other OS objects.

The following table gives the ROM and/or RAM requirements (in bytes) for each OS object in a simple configuration. Note that object sizes will vary depending on the project configuration and compiler packing issues.

Object	ROM	RAM
Alarm	2	12
Cat 2 ISR	8	0
Counter	20	4
CounterCallback	4	0
ExpiryPoint	3.5	0
OS Overheads (max)	0	69
OS-Application	0	0
PeripheralArea	0	0
Resource	8	4
ScheduleTable	16	12
Task	20	0

8.3 Stack Usage

The amount of stack used by each Task/ISR in RTA-OS is equal to the stack used in the Task/ISR body plus the context saved by RTA-OS. The size of the run-time context saved by RTA-OS depends on the Task/ISR type and the exact system configuration. The only reliable way to get the correct value for Task/ISR stack usage is to call the `Os_GetStackUsage()` API function.

Note that because RTA-OS uses a single-stack architecture, the run-time contexts of all tasks reside on the same stack and are recovered when the task terminates. As a result, run-time contexts of mutually exclusive tasks (for example, those that share an internal resource) are effectively overlaid. This means that the worst case stack usage can be significantly less than the sum of the worst cases of each object on the system. The RTA-OS tools automatically calculate the total worst case stack usage for you and present this as part of the configuration report.

8.4 Library Module Sizes

The RTA-OS kernel is demand linked. This means that each API call is placed into a separately linkable module. The following table lists the section sizes for each API module (in bytes) for the simple configuration in standard status.

Library Module	.bss	.os_exception_handlers	.os_exception_vectors	.os_primitives	.rodata	.text
ActivateTask						78
AdvanceCounter						4
CallTrustedFunction						22
CancelAlarm						50

Library Module	.bss	.os_exception_handlers	.os_exception_vectors	.os_primitives	.rodata	.text
ChainTask						70
CheckISRMemoryAccess						36
CheckObjectAccess						78
CheckObjectOwnership						74
CheckTaskMemoryAccess						36
ClearEvent						18
ControlIdle	4					28
DisableAllInterrupts	8					36
DispatchTask						120
ElapsedTime						98
EnableAllInterrupts						24
GetActiveApplicationMode						6
GetAlarm						90
GetAlarmBase						40
GetApplicationID						24
GetCounterValue						24
GetCurrentApplicationID						24
GetElapsedCounterValue						42
GetEvent						18
GetExecutionTime						18
GetISRID						6
GetIsrMaxExecutionTime						18
GetIsrMaxStackUsage						18
GetResource						40
GetScheduleTableStatus						24
GetStackSize						4
GetStackUsage						18
GetStackValue						10
GetTaskID						10
GetTaskMaxExecutionTime						18
GetTaskMaxStackUsage						18
GetTaskState						32
GetVersionInfo						24
Idle						4
InShutdown						2
IncrementCounter						14
InterruptSource					12	144

Library Module	.bss	.os_exception_handlers	.os_exception_vectors	.os_primitives	.rodata	.text
ModifyPeripheral						84
NextScheduleTable						80
Os_Cat1Wrappers		76				
Os_Cat2Wrappers		14				
Os_Cfg	36				776	116
Os_Cfg_Counters					472	3066
Os_Cfg_KL						36
Os_Exceptions		78				
Os_GetAbortStack	256					4
Os_GetCurrentIMask						6
Os_GetCurrentTPL						26
Os_H2T		84				
Os_IRQConst			120		1772	
Os_IRQHandler						44
Os_IRQSupport	4					4
Os_SVCHandler		32				
Os_StartCores	8					16
Os_Wrapper					16	84
Os_memclr						14
Os_mid_wrapper		92				
Os_setjmp				38		
Os_vec_init					40	154
ProtectionSupport						22
ReadPeripheral						66
ReleaseResource						44
ResetIsrMaxExecutionTime						18
ResetIsrMaxStackUsage						18
ResetTaskMaxExecutionTime						18
ResetTaskMaxStackUsage						18
ResumeAllInterrupts						24
ResumeOSInterrupts						32
Schedule						52
SetAbsAlarm						60
SetEvent						18
SetRelAlarm						96
SetScheduleTableAsync						36
ShutdownOS						42

Library Module	.bss	.os_exception_handlers	.os_exception_vectors	.os_primitives	.rodata	.text
StackOverrunHook						6
StartOS						170
StartScheduleTableAbs						78
StartScheduleTableRel						72
StartScheduleTableSynchron						36
StopScheduleTable						48
SuspendAllInterrupts	8					36
SuspendOSInterrupts	8					92
SyncScheduleTable						36
SyncScheduleTableRel						36
TerminateTask						14
ValidateCounter						34
ValidateISR						12
ValidateResource						28
ValidateScheduleTable						28
ValidateTask						28
WaitEvent						18
WritePeripheral						60

8.5 Execution Time

The following tables give the execution times in CPU cycles, i.e. in terms of ticks of the processor’s program counter. These figures will normally be independent of the frequency at which you clock the CPU. To convert between CPU cycles and SI time units the following formula can be used:

$$\text{Time in microseconds} = \text{Time in cycles} / \text{CPU Clock rate in MHz}$$

For example, an operation that takes 50 CPU cycles would be:

- at 20MHz = $50/20 = 2.5\mu s$
- at 80MHz = $50/80 = 0.625\mu s$
- at 150MHz = $50/150 = 0.333\mu s$

While every effort is made to measure execution times using a stopwatch running at the same rate as the CPU clock, this is not always possible on the target hardware. If the stopwatch runs slower than the CPU clock, then when RTA-OS reads the stopwatch, there is a possibility that the time read is less than the actual amount of time that has elapsed due to the difference in resolution between the CPU clock and the stopwatch (the *User Guide* provides further details on the issue of uncertainty in execution time measurement).

The figures presented in Section 8.5.1 have an uncertainty of 0 CPU cycle(s).

8.5.1 Context Switching Time

Task switching time is the time between the last instruction of the previous task and the first instruction of the next task. The switching time differs depending on the switching contexts (e.g. an `ActivateTask()` versus a `ChainTask()`).

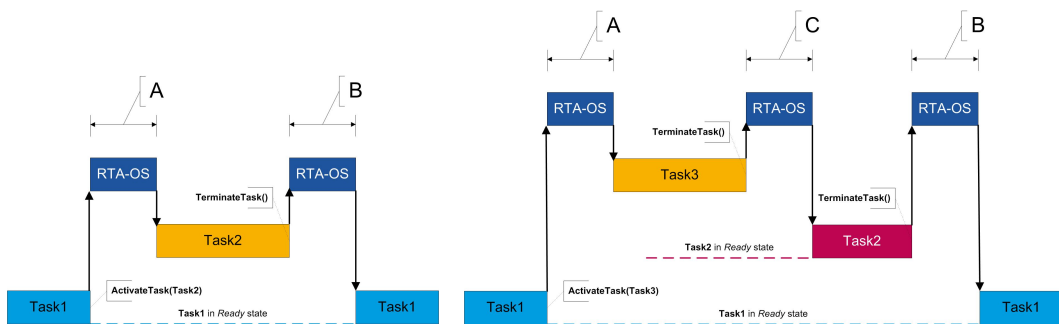
Interrupt latency is the time between an interrupt request being recognized by the target hardware and the execution of the first instruction of the user provided handler function:

For Category 1 ISRs this is the time required for the hardware to recognize the interrupt.

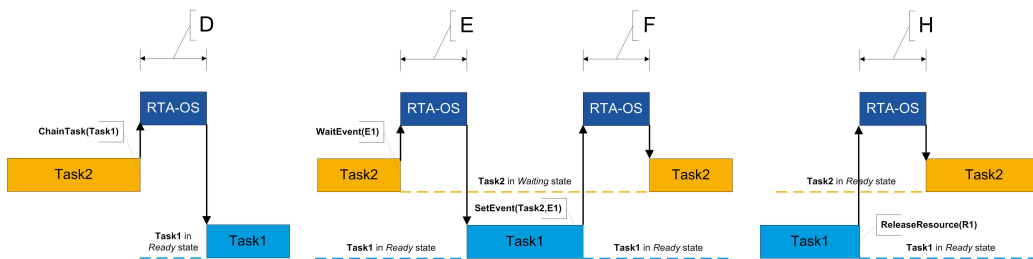
For Category 2 ISRs this is the time required for the hardware to recognize the interrupt plus the time required by RTA-OS to set-up the context in which the ISR runs.

Figure 8.1 shows the measured context switch times for RTA-OS.

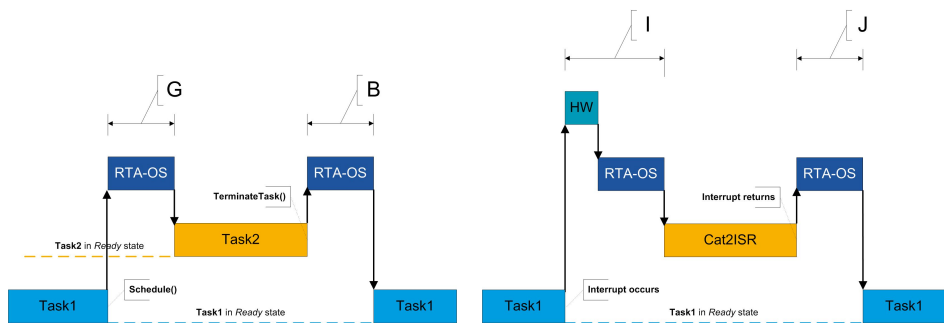
Switch	Key	CPU Cycles	Actual Time
Task activation	A	283	1.13us
Task termination with resume	B	80	320ns
Task termination with switch to new task	C	197	788ns
Chaining a task	D	397	1.59us
Waiting for an event resulting in transition to the WAITING state	E	705	2.82us
Setting an event results in task switch	F	1043	4.17us
Non-preemptive task offers a preemption point (co-operative scheduling)	G	292	1.17us
Releasing a resource results in a task switch	H	276	1.1us
Entering a Category 2 ISR	I	410	1.64us
Exiting a Category 2 ISR and resuming the interrupted task	J	286	1.14us
Exiting a Category 2 ISR and switching to a new task	K	417	1.67us
Entering a Category 1 ISR	L	155	620ns



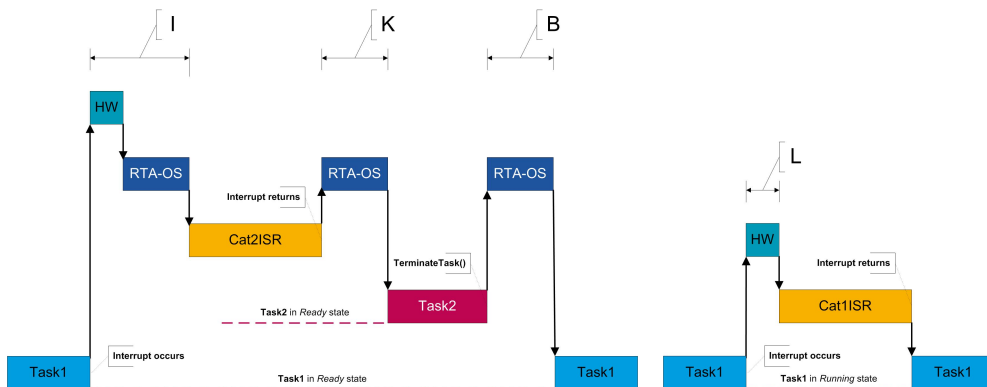
(a) Task activated. Termination resumes preempted task. (b) Task activated. Termination switches into new task.



(c) Task chained. (d) Task waits. Task is resumed when event set. (e) Task switch when resource is released.



(f) Request for scheduling made by non-preemptive task. (g) Category 2 interrupt entry. Interrupted task resumed on exit.



(h) Category 2 interrupt entry. Switch to new task on exit. (i) Category 1 interrupt entry.

Figure 8.1: Context Switching

9 Finding Out More

Additional information about TraveoII/GHS-specific parts of RTA-OS can be found in the following manuals:

TraveoII/GHS Release Note. This document provides information about the TraveoII/GHS port plug-in release, including a list of changes from previous releases and a list of known limitations.

Information about the port-independent parts of RTA-OS can be found in the following manuals, which can be found in the RTA-OS installation (typically in the Documents folder):

Getting Started Guide. This document explains how to install RTA-OS tools and describes the underlying principles of the operating system

Reference Guide. This guide provides a complete reference to the API, programming conventions and tool operation for RTA-OS.

User Guide. This guide shows you how to use RTA-OS to build real-time applications.

10 Contacting ETAS

10.1 Technical Support

Technical support is available to all users with a valid support contract. If you do not have a valid support contract, please contact your regional sales office (see below).

The best way to get technical support is by email. Any problems or questions about the use of the product should be sent to:

rta.hotline@etas.com

If you prefer to discuss your problem with the technical support team, you call the support hotline on:

+44 (0)1904 562624.

The hotline is available during normal office hours (0900-1730 GMT/BST).

In either case, it is helpful if you can provide technical support with the following information:

- Your support contract number
- The version of the ETAS tools you are using
- The version of the compiler tool chain you are using
- The command line (or reproduction of steps) that result in an error message
- The error messages or return codes you received (if any)
- Your .xml, .arxml and .rtaos files
- The file Diagnostic.dmp if it was generated

10.2 General Enquiries

10.2.1 ETAS Global Headquarters

ETAS GmbH

Borsigstrasse 24
70469 Stuttgart
Germany

Phone:	+49 711 3423-0
Fax:	+49 711 3423-2106
WWW:	www.etas.com

10.2.2 ETAS Local Sales & Support Offices

Contact details for your local sales office and local technical support team (where available) can be found on the ETAS web site:

ETAS subsidiaries	www.etas.com/en/contact.php
ETAS technical support	www.etas.com/en/hotlines.php

Index

A

- Assembler, [45](#)
- AUTOSAR OS includes
 - Os.h, [29](#)
 - Os_Cfg.h, [29](#)
 - Os_MemMap.h, [29](#)

C

- CAT1_ISR, [39](#)
- Compiler, [44](#)
- Compiler (Green Hills Software, Compiler v2017.1.4), [43](#)
- Compiler (Green Hills Software, Compiler v2019.1.4), [43](#)
- Compiler Versions, [43](#)
- Configuration
 - Port-Specific Parameters, [21](#)

D

- Debugger, [47](#)

E

- ETAS License Manager, [11](#)
 - Installation, [11](#)

F

- Files, [28](#)

H

- Hardware
 - Requirements, [9](#)

I

- Installation, [9](#)
 - Default Directory, [10](#)
 - Verification, [19](#)
- Interrupts, [52](#)
 - Category 1, [54](#)
 - Category 2, [55](#)
 - Category 2 Software Interrupts, [55](#)
 - Default, [55](#)
- IPL, [53](#)

L

- Librarian, [46](#)
- Library
 - Name of, [29](#)

License, [11](#)

- Borrowing, [15](#)
- Concurrent, [12](#)
- Grace Mode, [12](#)
- Installation, [15](#)
- Machine-named, [12](#)
- Status, [15](#)
- Troubleshooting, [16](#)
- User-named, [12](#)

Linker, [46](#)

M

- Memory Model, [55](#)

O

- Options, [44](#)
- Os_AwaitStartup, [30](#)
- Os_Cbk_GetAbortStack, [32](#)
- Os_Cbk_GetSetProtection, [33](#)
- Os_Cbk_SetMemoryAccess, [34](#)
- Os_Cbk_StartCore, [38](#)
- Os_Clear_x, [40](#)
- Os_ClearDemuxInterrupts, [40](#)
- Os_Disable_x, [41](#)
- Os_DisableAllConfiguredInterrupts, [40](#)
- Os_Enable_x, [41](#)
- Os_EnableAllConfiguredInterrupts, [41](#)
- Os_InitializeVectorTable, [31](#)
- Os_IntChannel_x, [42](#)
- Os_StackSizeType, [42](#)
- Os_StackValueType, [42](#)

P

- Parameters of Implementation, [21](#)
- Performance, [57](#)
 - Context Switching Times, [62](#)
 - Library Module Sizes, [58](#)
 - RAM and ROM, [57](#)
 - Stack Usage, [58](#)
- Processor Modes, [55](#)
 - Trusted, [55](#)
 - Untrusted, [55](#)
- Processor state when calling StartOS(), [56](#)

R

- Registers

- ACTLR, [52](#)
- BASEPRI, [52](#)
- CCR, [52](#)
- CCR.NONBASETHRDENA, [51](#)
- CCR.STKALIGN, [51](#)
- CONTROL, [52](#)
- CONTROL.FPCA, [51](#)
- CONTROL.nPRIV, [51](#)
- CONTROL.SPSEL, [51](#)
- CPACR, [51](#), [52](#)
- CPUSS_CMx_SYSTEM_INT_CTL, [52](#)
- FAULTMASK, [51](#), [52](#)
- FPCCR, [51](#), [52](#)
- Initialization, [50](#)
- IPSR, [51](#)
- Non-modifiable, [51](#)
- NVIC, [51](#), [52](#)
- PRIMASK, [51](#), [52](#)
- SHCSR, [52](#)
- SHPRx, [51](#), [52](#)
- SP (R13), [51](#), [52](#)
- VTOR, [51](#)
- Resource
 - BusFault, [52](#)
 - HardFault, [52](#)
 - MemManage, [52](#)
 - SVCall, [52](#)
 - UsageFault, [52](#)
- S**
 - Software
 - Requirements, [9](#)
 - Spurious interrupts when termination an OS Application, [56](#)
 - Stack, [56](#)
- T**
 - Target, [49](#)
 - Variants, [50](#)
 - Toolchain, [43](#)
- U**
 - Use of the Inter-Processor Communication hardware in multicore applications, [56](#)
- V**
 - Variants, [50](#)
 - Vector Table
 - Base Address, [54](#)