RTA-OS PPCe200/WR V5.1.25

Port Guide

Status: Released

DRIVING | EMBEDDED EXCELLENCE

Copyright

Document: 10596-PG-5.1.25 EN-07-2019

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 ORTI/Lauterbach . . . 23
4.2.7 ORTI/winIDEA . . . 23
4.2.8 ORTI Stack Fill . . . 23
4.2.9 Support winIDEA Analyzer . . . 23
4.2.10 Link Type . . . 23
4.2.11 SDA RAM Threshold . . . 24
4.2.12 SDA ROM Threshold . . . 24
4.2.13 MultiCore Lock . . . 24
4.2.14 OS Locks disable Cat1 . . . 25
4.2.15 MultiCore interrupts . . . 25
4.2.16 Preserve SPE . . . 25
4.2.17 Enable stack repositioning . . . 26
4.2.18 Enable untrusted stack check . . . 26
4.2.19 Use software vectoring . . . 26
4.2.20 Block default interrupt . . . 26
4.2.21 Generate Cat1 EOIR . . . 27
4.2.22 Cached CoreID register . . . 27

4.2.23 Always call GetAbortStack 27
4.2.24 Use Short Enum 28
4.2.25 Customer Option 1 28
4.3 Generated Files 29

5 Port-Specific API 30
5.1 API Calls 30
5.1.1 Os_CacheCoreID 30
5.1.2 Os_InitializeVectorTable 31
5.2 Callbacks 31
5.2.1 Os_Cbk_GetAbortStack 31
5.2.2 Os_Cbk_StartCore 32
5.3 Macros 33
5.3.1 CAT1_ISR 33
5.3.2 Os_DisableAllConfiguredInterrupts 33
5.3.3 Os_Disable_x 34
5.3.4 Os_EnableAllConfiguredInterrupts 34
5.3.5 Os_Enable_x 34
5.3.6 Os_IntChannel_x 34
5.4 Type Definitions 35
5.4.1 Os_StackSizeType 35
5.4.2 Os_StackValueType 35

6 Toolchain 36
6.1 Compiler Versions 36
6.1.1 WindRiver (Diab) v5.8.0.0 patch 5 (MPC5676R/MPC5777C only) 36
6.1.2 WindRiver (Diab) v5.8.0.0 patch 6 (MPC5676R/MPC5777C only) 36
6.1.3 WindRiver (Diab) v5.9.4.0 36
6.1.4 WindRiver (Diab) v5.9.4.7 37
6.1.5 WindRiver (Diab) v5.9.6.1 37
6.1.6 WindRiver (Diab) v5.9.6.2 37
6.1.7 WindRiver (Diab) v5.9.6.6 37
6.2 Options used to generate this guide 37
6.2.1 Compiler 37
6.2.2 Assembler 38
6.2.3 Librarian 39
6.2.4 Linker 39
6.2.5 Debugger 40

7 Hardware **42**
7.1 Supported Devices . . . 42
7.2 Register Usage . . . 45
7.2.1 Initialization . . . 45
7.2.2 Modification . . . 46
7.3 Required OS resources . . . 46
7.3.1 Core ID Caching . . . 46
7.4 Interrupts . . . 46
7.4.1 Interrupt Priority Levels . . . 47
7.4.2 Allocation of ISRs to Interrupt Vectors . . . 47
7.4.3 Vector Table . . . 48
7.4.4 Using Raw Exception Handlers . . . 49
7.4.5 Writing Category 1 Interrupt Handlers . . . 49
7.4.6 Writing Category 2 Interrupt Handlers . . . 49
7.4.7 Default Interrupt . . . 50
7.5 Memory Model . . . 50
7.6 Processor Modes . . . 50
7.7 Stack Handling . . . 50

8 Performance **51**
8.1 Measurement Environment . . . 51
8.2 RAM and ROM Usage for OS Objects . . . 51
8.2.1 Single Core . . . 52
8.2.2 Multi Core . . . 52
8.3 Stack Usage . . . 52
8.4 Library Module Sizes . . . 53
8.4.1 Single Core . . . 53
8.4.2 Multi Core . . . 55
8.5 Execution Time . . . 58
8.5.1 Context Switching Time . . . 59

9 Finding Out More **61**

10 Contacting ETAS **62**
10.1 Technical Support . . . 62
10.2 General Enquiries . . . 62
10.2.1 ETAS Global Headquarters . . . 62
10.2.2 ETAS Local Sales & Support Offices . . . 62

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 -> R4.3.1) 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 PPCe200/WR port plug-in that customizes the RTA-OS development tools for the Freescale/ST MPC5xxx/SPC5xx with the WindRiver (Diab) 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 PPCe200/WR port and cover:

- how to install the PPCe200/WR port plug-in;
- how to configure PPCe200/WR-specific attributes;
- how to build an example application to check that the PPCe200/WR 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 MPC5xxx/SPC5xx, 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:

`http://www.osek-vdx.org`

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 PPCe200/WR 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 7
- 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* v4.0 *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 *v4.0* 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 PPCe200/WR 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 PPCe200/WR 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 `PPCe200WR_5.1.25`. This contains everything to do with the port plug-in.

Each version of the port installs in its own directory - the trailing `_5.1.25` 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:

PPCe200WR.dll - the port plug-in that is used by **rtaosgen** and **rtaoscfg**.

RTA-OS PPCe200WR Port Guide.pdf - the documentation for the port (the document you are reading now).

RTA-OS PPCe200WR 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`.

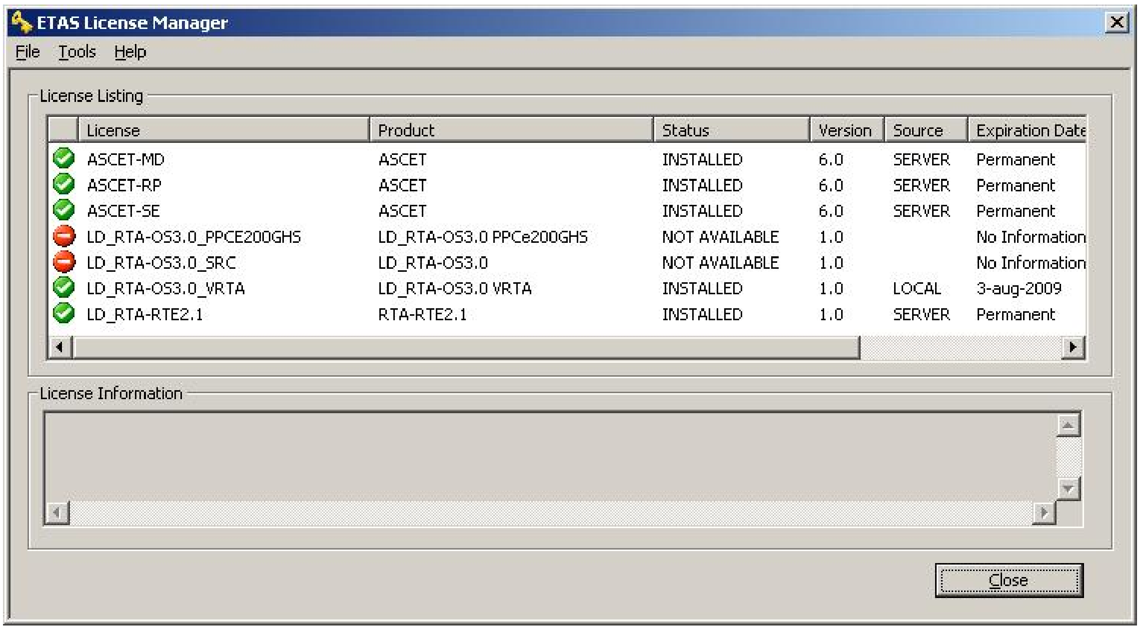


Figure 2.1: The ETAS License manager

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 username 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.

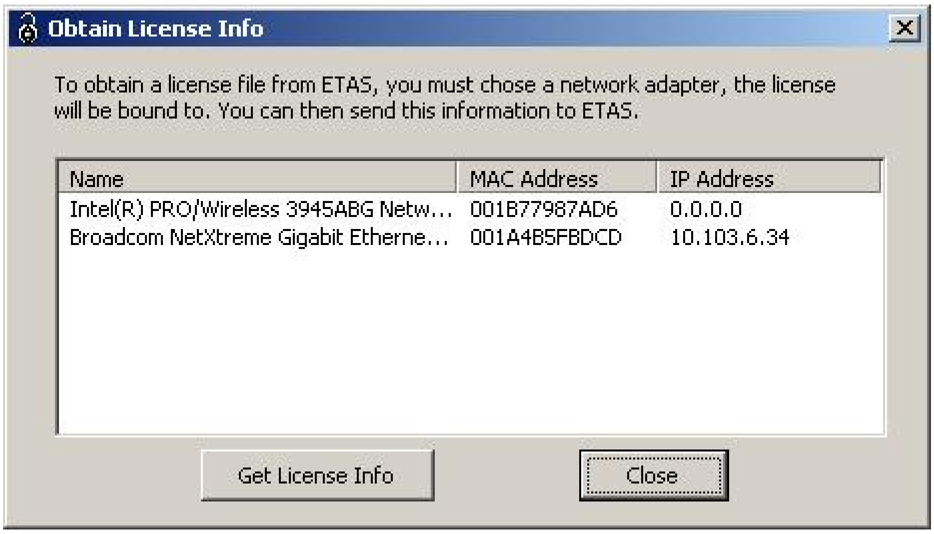


Figure 2.2: Obtaining License Information

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.

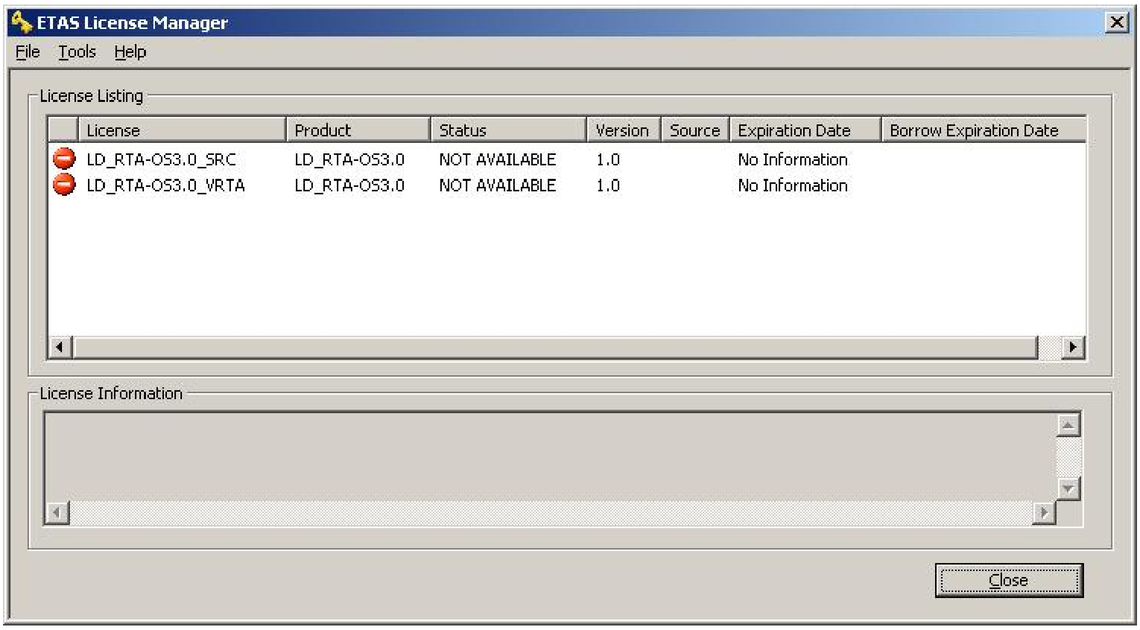


Figure 2.3: Unlicensed RTA-OS Installation

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”

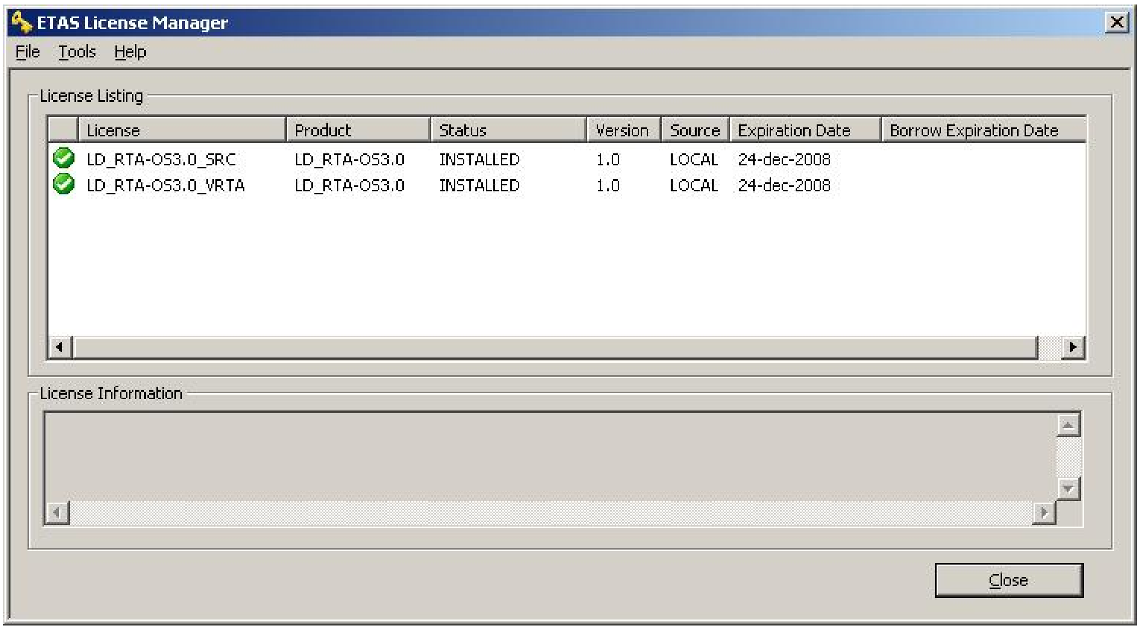


Figure 2.4: Licensed features for RTA-OS

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`.

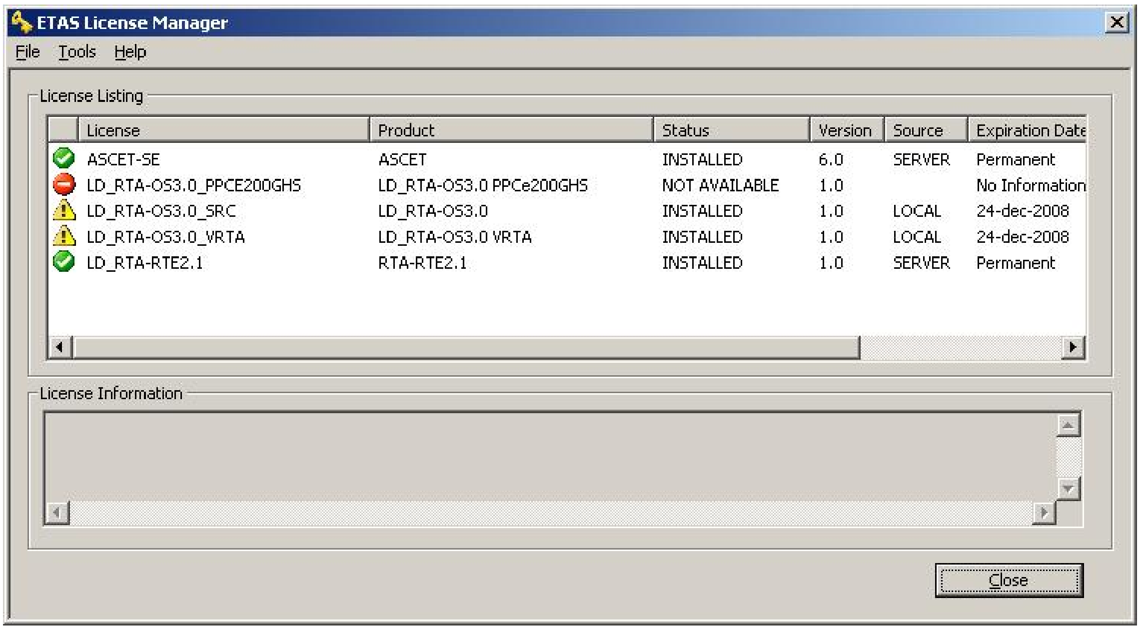


Figure 2.5: Licensed features that are due to expire

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 PPCe200/WR 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 PPCe200/WR 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 PPCe200/WR 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.

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 ORTI/Lauterbach

XML name Orti22Lauterbach

Description

Enables ORTI generation for Lauterbach debugger.

4.2.7 ORTI/winIDEA

XML name Orti21winIDEA

Description

Enables ORTI generation for winIDEA debugger.

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.

4.2.9 Support winIDEA Analyzer

XML name winIDEAAnalyzer

Description

Adds support for the winIDEA profiler to track ORTI items. Context switches take a few cycles longer as additional code is inserted to support this feature.

4.2.10 Link Type

XML name OSLinkMemModel

Description

Select the type of map used in linker samples.

Settings

Value	Description
IntRAM	Code/data in internal RAM (default)
IntFLASH	Code in internal flash, data in internal RAM

4.2.11 SDA RAM Threshold

XML name sda_value

Description

Sets the value used for small data objects when compiling. Defaults to zero.

4.2.12 SDA ROM Threshold

XML name sda_value_const

Description

Sets the value used for small const objects when compiling. Defaults to zero.

4.2.13 MultiCore Lock

XML name MC_Locker

Description

Select hardware used for Spinlock implementation. The Software option is only applicable to select SPC57x/SPC58x/MPC57xx variants. The default option is Software where supported.

Settings

Value	Description
Software	Software decorated instruction (default)
SEMA4_Gate00	SEMA4 Gate00
SEMA4_Gate01	SEMA4 Gate01
SEMA4_Gate02	SEMA4 Gate02
SEMA4_Gate03	SEMA4 Gate03
SEMA4_Gate04	SEMA4 Gate04
SEMA4_Gate05	SEMA4 Gate05
SEMA4_Gate06	SEMA4 Gate06
SEMA4_Gate07	SEMA4 Gate07
SEMA4_Gate08	SEMA4 Gate08
SEMA4_Gate09	SEMA4 Gate09
SEMA4_Gate10	SEMA4 Gate10
SEMA4_Gate11	SEMA4 Gate11
SEMA4_Gate12	SEMA4 Gate12
SEMA4_Gate13	SEMA4 Gate13
SEMA4_Gate14	SEMA4 Gate14
SEMA4_Gate15	SEMA4 Gate15

4.2.14 OS Locks disable Cat1

XML name OSLockDisableAll

Description

Specify whether all interrupts are disabled while internal OS spinlocks are held. This does not affect spinlocks accessed using the GetSpinlock or TryToGetSpinlock APIs

Settings

Value	Description
true	Disable all interrupts
false	Do not disable interrupts (default)

4.2.15 MultiCore interrupts

XML name MC_Interrupt

Description

Select the first software interrupt to use for multi-core implementation. The OS will use the appropriate number of consecutive interrupts.

Settings

Value	Description
0	INTC0 (default)
1	INTC1
2	INTC2
3	INTC3
4	INTC4
5	INTC5
6	INTC6

4.2.16 Preserve SPE

XML name preserve_spe

Description

Select whether SPE related registers are preserved across TASK and Category 2 ISR preemptions. The number of SPE related registers varies between variants and cores. Registers are only preserved if the MSR[SPE] bit is set. Application code should normally initialize the SPE bit as appropriate for each core before calling StartOS() and not modify it subsequently.

Settings

Value	Description
true	Preserve SPE related registers
false	Do not preserve SPE related registers (default)

4.2.17 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)

4.2.18 Enable untrusted stack check

XML name DistrustStacks

Description

Extra code can be placed in interrupt handlers to detect when untrusted code has an illegal stack pointer value. Also exception handlers run on a private stack (Refer to the documentation for Os_Cbk_GetAbortStack). This has a small performance overhead, so is made optional.

Settings

Value	Description
true	Perform the checks (default)
false	Do not check

4.2.19 Use software vectoring

XML name SoftwareVectoring

Description

Select software-based dispatching of interrupts. RTA-OS will provide the software dispatching code unless you specify your own dispatcher by configuring an interrupt on IVOR 4.

Settings

Value	Description
true	Software vectoring
false	Hardware vectoring (default)

4.2.20 Block default interrupt

XML name block_default_interrupt

Description

Where a default interrupt is specified, it will normally execute if a spurious interrupt fires. You can block this behavior using this option. The option affects the priority assigned to unused interrupt sources.

Settings

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

4.2.21 Generate Cat1 EOIR

XML name GenerateEOIR

Description

When hardware vector mode is used, each Category 1 ISR must signal the end of interrupt by writing to the EOIR register in the interrupt controller. If this target option is set to TRUE, RTA-OS will generate the correct EOIR code for you as part of the CAT1_ISR macro. In software vectoring mode RTA-OS must write the EOIR in its interrupt dispatcher, so Category 1 ISRs must not write to EOIR. In software vectoring mode you should leave this option undefined or FALSE.

Settings

Value	Description
true	EOIR code is added to CAT1_ISR
false	Code EOIR code is not added (default)

4.2.22 Cached CoreID register

XML name CachedCoreID

Description

Specify the register to cache the AUTOSAR core ID into. This is necessary for multi-core configurations. It defaults to either SPRG4 or PMGC0 (variant dependent).

Settings

Value	Description
SPRG4	SPRG4
SPRG5	SPRG5
SPRG6	SPRG6
SPRG7	SPRG7
PMGC0	PMGC0

4.2.23 Always call GetAbortStack

XML name Always_call_GetAbortStack

Description

When an exception or mem exception 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)

4.2.24 Use Short Enum

XML name use_short_enum

Description

Set to enable the use of short enums.

Settings

Value	Description
true	Enables short enum
false	Disables short enum (default)

4.2.25 Customer Option 1

XML name customer_option1

Description

This option activates certain optimizations to the code that are only valid for a specific set of conditions. In particular the APIs SuspendAllInterrupts, ResumeAllInterrupts, SuspendOSInterrupts, ResumeOSInterrupts, DisableAllInterrupts, EnableAllInterrupts and Os_GetExecutionTime are optimized for speed. For this option to be valid, the following conditions must be met: 1: Untrusted code must be allowed to modify INTC registers, 2: Untrusted code must be allowed to read and write OS data. Note also that DisableAllInterrupts() will not disable any interrupt with priority 16, because it will not modify the EE-bit. It should only be enabled with agreement from ETAS.

Settings

Value	Description
true	Enables Customer Option 1
false	Standard behavior (default)

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 PPCe200/WR 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_CacheCoreID

Used to cache the AUTOSAR core ID into register SPRG4-7 or PMGC0.

Syntax

```
FUNC(void, OS_APPL_CODE) Os_CacheCoreID(void)
```

Description

In multi-core configurations it is necessary to cache the AUTOSAR core ID into a register that can be read efficiently by trusted and untrusted code. The target option 'Cached CoreID register' is used to specify which register to use from one of SPRG4, SPRG5, SPRG6, SPRG7 or PMGC0. The default is either SPRG4 or PMGC0 (variant dependent). The Os_CacheCoreID() must be called to do this caching for you. It must be called on each core before any other OS call is made (including GetCoreID()). If you use the OS_MAIN() macro, then this will silently call Os_CacheCoreID() for you. Similarly it will be called during the execution of Os_InitializeVectorTable().

Example

```
OS_MAIN() {
  /* The OS_MAIN macro implicitly calls Os_CacheCoreID() */
  ...
}

or

void main(void) {
  Os_CacheCoreID();
  ...
}

or

void main(void) {
  Os_InitializeVectorTable(); /* Os_InitializeVectorTable calls
      Os_CacheCoreID() */
  ...
```

```
}
```

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

5.1.2 Os_InitializeVectorTable

Initialize the interrupt hardware and vector table(s).

Syntax

```
void Os_InitializeVectorTable(void)
```

Description

Os_InitializeVectorTable() initializes the CPU and INTC interrupts according to the requirements of the project configuration. In particular, it sets IVPR, IVOR registers and INTC priorities. It sets hardware or software vectoring mode based on whether IVOR_4 has been assigned to an ISR.

Os_InitializeVectorTable() should be called before StartOS(). It should be called even if ’Suppress Vector Table Generation’ is set to TRUE.

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 *, {memclass}) 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 untrusted code's stack pointer is invalid. For this reason, if 'Enable untrusted stack check' 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. Maskable interrupts will be disabled during this process so the stack only needs to be large enough to get to and execute ProtectionHook. A default implementation of Os_Cbk_GetAbortStack is supplied in the RTA-OS library that returns the address of an area of static memory, but you can implement your own version to override its behavior.

Note: memclass is OS_APPL_CODE for AUTOSAR 3.x, OS_CALLOUT_CODE for AUTOSAR 4.0, OS_OS_CBK_GETABORTSTACK_CODE for AUTOSAR 4.1.

Example

```
FUNC(void *, {memclass}) Os_Cbk_GetAbortStack(void) {
  /* Could be implemented to return a core-specific location.
   * The last location of the array is treated as the previous
   * stack frame where the LR would be stored */
  static uint32 abortstack[40U];
  return &abortstack[38U];
}
```

Required when

The callback must be present if 'Enable untrusted stack check' is configured and there are untrusted OS Applications. The callback is also present if the 'Always call GetAbort-Stack' target option is enabled.

5.2.2 Os_Cbk_StartCore

Callback routine used to start a non master core on a multicore 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 and 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 and causing it to enter OS_MAIN.

RTA-OS provides a default implementation of Os_Cbk_StartCore that sets the core reset vector to ’os_example_init_core’ and then releases the core.

Os_Cbk_StartCore does not get called for core 0, because core 0 must start first.

Note: memclass is OS_APPL_CODE for AUTOSAR 3.x, OS_CALLOUT_CODE for AUTOSAR 4.0, OS_OS_CBK_STARTCORE_CODE for AUTOSAR 4.1.

Example

```
FUNC(StatusType, {memclass}) Os_Cbk_StartCore(uint16 CoreID) {
  SET_CORE_RSTVEC(CoreID);
  RELEASE_CORE(CoreID);
}
```

Required when

Required for non master cores that will be started.

See Also

StartCore
StartNonAutosarCore
StartOS

5.3 Macros

5.3.1 CAT1_ISR

Macro that should be used to create a Category 1 ISR entry function. This macro exists to help make your code portable between targets.

Example

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

5.3.2 Os_DisableAllConfiguredInterrupts

The Os_DisableAllConfiguredInterrupts macro will disable all configured INTC interrupts by adjusting the INTC PSR settings. You will need to #include the file "Os_DisableInterrupts.h" if you want to use this macro. It may not be used by untrusted code.

Example

```
Os_DisableAllConfiguredInterrupts()
Os_Enable_Millisecond()
```

5.3.3 Os_Disable_x

The Os_Disable_x macro will disable the named INTC interrupt vector by adjusting its INTC PSR settings. The macro can be called using either the INTC vector name or the RTA-OS configured vector name. In the example, this is Os_Disable_eMIOS_Channel_0() and Os_Disable_Millisecond() respectively. You will need to #include the file "Os_DisableInterrupts.h" if you want to use this macro. It may not be used by untrusted code.

Example

```
Os_Disable_eMIOS_Channel_0()
Os_Disable_Millisecond()
```

5.3.4 Os_EnableAllConfiguredInterrupts

The Os_EnableAllConfiguredInterrupts macro will enable all configured INTC interrupts by adjusting the INTC PSR settings. You will need to #include the file "Os_DisableInterrupts.h" if you want to use this macro. It may not be used by untrusted code.

Example

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

5.3.5 Os_Enable_x

The Os_Enable_x macro will re-enable the named INTC interrupt vector at the priority it was configured with by adjusting its INTC PSR settings. The macro can be called using either the INTC vector name or the RTA-OS configured vector name. In the example, this is Os_Enable_eMIOS_Channel_0() and Os_Enable_Millisecond() respectively. You will need to #include the file "Os_DisableInterrupts.h" if you want to use this macro. It may not be used by untrusted code.

Example

```
Os_Enable_eMIOS_Channel_0()
Os_Enable_Millisecond()
```

5.3.6 Os_IntChannel_x

The Os_IntChannel_x macro can be used to get the vector number associated with the named INTC interrupt (0, 1, 2...). The macro can be called using either the INTC vector name or the RTA-OS configured vector name. In the example, this is Os_IntChannel_eMIOS_Channel_0 and Os_IntChannel_Millisecond respectively. You will need to #include the file "Os_DisableInterrupts.h" if you want to use this macro.

Example

```
trigger_interrupt(Os_IntChannel_eMIOS_Channel_0);
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 (ESP).

Example

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

6 Toolchain

This chapter contains important details about RTA-OS and the WindRiver (Diab) 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.*

6.1 Compiler Versions

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

6.1.1 WindRiver (Diab) v5.8.0.0 patch 5 (MPC5676R/MPC5777C only)

Ensure that dcc.exe is on the path.

Tested on For this release this version has not been tested.

6.1.2 WindRiver (Diab) v5.8.0.0 patch 6 (MPC5676R/MPC5777C only)

Ensure that dcc.exe is on the path.

Tested on The release tests were performed on this version.

6.1.3 WindRiver (Diab) v5.9.4.0

Ensure that dcc.exe is on the path.

Tested on The release tests were performed on this version.

6.1.4 WindRiver (Diab) v5.9.4.7

Ensure that dcc.exe is on the path.

Tested on The release tests were performed on this version.

6.1.5 WindRiver (Diab) v5.9.6.1

Ensure that dcc.exe is on the path.

Tested on The release tests were performed on this version.

6.1.6 WindRiver (Diab) v5.9.6.2

Ensure that dcc.exe is on the path.

Tested on The release tests were performed on this version.

6.1.7 WindRiver (Diab) v5.9.6.6

Ensure that dcc.exe is on the path.

Tested on The release tests were performed on this version.

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 dcc.exe
Version Rel 5.9.6.6

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.

`-Xforce-prototypes` Reports an error for functions with no prototype

`-Xstop-on-warning` Treat all warnings as errors

`-XO` Enable extra optimizations

`-ei1824` Remove discards volatile qualifier warning (possible bug in 5.9.4.7 compiler)

`-Xwhole-program-optim=0` Disable whole-program-optimization on the RTA-OS library

`-Xdebug-local-cie` Generate a local Common Information Entry (CIE) for each unit

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.

`-Xnested-interrupts` Allows nested interrupts (supports the CAT1_ISR macro)

`-Xsmall-data=8` SDA data threshold (value set by target option)

`-Xsmall-const=0` SDA const threshold (value set by target option)

`-tPPCE200Z425N3VES:simple` Generate code for target processor (variant-specific). Application code can use the VEG version of this option if it needs to use hardware floating point. This is compatible with a VES-built OS.

`-Xapu-volatile-context` Use the Volatile Context Save/Restore APU instructions

`-Xpragma-section-first` Compiler uses earliest currently valid section pragma that specifies a non-default location for the function or variable

`-Xenum-is-int` Disable short enum (value set by target option)

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

6.2.2 Assembler

Name dcc.exe
Version Rel 5.9.6.6

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 dar.exe
Version Rel 5.9.6.6

6.2.4 Linker

Name dld.exe
Version Rel 5.9.6.6

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.

`-tPPCE200Z425N3VES:simple` Generate code for target processor (variant-specific). Application code can use the VEG version of this option if it needs to use hardware floating point. This is compatible with a VES-built OS.

`-m14` Produce detailed map file

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 104140 or later

Notes

Supports .elf files and ORTI files.

Notes on using ORTI with the debugger

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). For a single core system (i.e. core 0 only) the labels are:

```
OS_STACK0_BASE = ADDR(stackcore0);
OS_STACK0_SIZE = SIZEOF(stackcore0);
```

where stackcore0 is the section containing the Core 0 stack.

The fill pattern used by the debugger must be contained with in 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.

ORTI with the winIDEA debugger

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

ORTI Stack Fill with the winIDEA debugger

Again the 'ORTI Stack Fill' target option is provided to extend the ORTI support to allow evaluation of unused stack space. The stack use is then displayed in the 'Operating System' window in addition to the other ORTI information. 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). For a single core system (i.e. core 0 only) the labels are:

```
OS_STACK0_BASE = ADDR(stackcore0);
OS_STACK0_SIZE = SIZEOF(stackcore0);
```

where stackcore0 is the section containing the Core 0 stack.

The application must contain 32 bit constant values referencing these labels to allow the debugger to visualize these.

```
extern const uint32 OS_STACK0_BASE;
extern const uint32 OS_STACK0_SIZE;
const uint32 Os_Stack0_Start = (const uint32)&OS_STACK0_BASE;
const uint32 Os_Stack0_Size = (const uint32)&OS_STACK0_SIZE;
```

The fill pattern used by the debugger is set to 0xCAFEF00D by default in the ORTI file. If a different fill pattern is required then the ORTI file must be edited.

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

Using the winIDEA Analyzer

When profiler analyzer support for the winIDEA debugger is enabled the entry and exit times for Tasks and Category 2 interrupts are increased by a few cycles to allow measurement of these objects.

7 Hardware

7.1 Supported Devices

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

Name: Freescale/ST
Device: MPC5xxx/SPC5xx

The following variants of the MPC5xxx/SPC5xx are supported:

- MPC5534
- MPC5561
- MPC5565
- MPC5566
- MPC5567
- MPC5604B
- MPC5604C
- MPC5604E
- MPC5604S
- MPC5605B
- MPC5606B
- MPC5606BK
- MPC5607B
- MPC5633
- MPC5642A
- MPC5643L
- MPC5644B
- MPC5644C
- MPC5645B
- MPC5645C
- MPC5645S
- MPC5646B
- MPC5646C

- MPC5673Fv2
- MPC5674Fv2
- MPC5675K
- MPC5676R
- MPC5726L
- MPC5726L_JDP
- MPC5744Kv2
- MPC5744Kv2_JDP
- MPC5744P
- MPC5745B
- MPC5745R
- MPC5745Rv2
- MPC5746B
- MPC5746C
- MPC5746Gv2
- MPC5746Mv2
- MPC5746Mv2_JDP
- MPC5746R
- MPC5746Rv2
- MPC5747Cv2
- MPC5748GCompatibility
- MPC5748Gv2
- MPC5775K
- MPC5777C
- MPC5777Mv2
- MPC5777Mv2_JDP
- S32R274
- SPC560B40
- SPC560B44

- SPC560B50
- SPC560B54
- SPC560B60
- SPC560B64
- SPC560C
- SPC560P
- SPC560S
- SPC563M
- SPC56EL70
- SPC56HK70
- SPC570S40
- SPC572L64
- SPC572L64_JDP
- SPC574K72v2
- SPC574K72v2_JDP
- SPC574S60
- SPC582B50
- SPC582B54
- SPC582B60
- SPC584B
- SPC584C70
- SPC584C74
- SPC58EC70
- SPC58EC70_JDP
- SPC58EC74
- SPC58EC74_JDP
- SPC58EC80
- SPC58EC80_JDP
- SPC58EG80

- SPC58EG84
- SPC58NE84v2
- SPC58NE84v2_JDP
- SPC58NH92
- SPC58NN84
- SPC58NN84_JDP

If you require support for a variant of MPC5xxx/SPC5xx 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
CCU	Multicore: The Cache Coherency Unit must be enabled (when present).
INTC_PSRx	The INTC priorities have to be set to the values declared in the configuration. This can be done by calling Os_InitializeVectorTable().
IVORx	The IVOR vectors have to be set correctly based on the configuration. This can be done by calling Os_InitializeVectorTable().
IVPR	The interrupt base address has to be set to the start of Os_InterruptVectorTable. This can be done by calling Os_InitializeVectorTable().
L1CSR	Multicore: The instruction cache may be enabled. The data cache must be set to write-through if enabled.
MSR	MSR[EE] should be set and MSR[PR] must be reset. The MSR[SPE] must only be set on cores where you intend to use SPE features.
SPRG4-7/PMGC0	Multicore: Register used to store the cached core ID. This can be done by calling Os_CacheCoreID() on each core before calling any RTA-OS API, including StartOS().
TLB	The INTC must be cache-inhibited and guarded (where appropriate).
XBAR	Multicore: Round-robin scheduling is needed for core accesses to ROM and RAM from a code execution perspective.

7.2.2 Modification

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

Register	Notes
INTC_BCR	INTC configuration register. (Or equivalent for Multicore.)
INTC_CPR	INTC priority register. (Or equivalent for Multicore.)
INTC_EOIR	INTC end of interrupt register. (Or equivalent for Multicore.) If the 'Generate Cat1 EOIR' target option is disabled or if raw interrupts are used then it is permitted to modify this register.
INTC_IACKR	INTC acknowledge register. (Or equivalent for Multicore.)
INTC_PSRx	INTC priority select register.
IVPR	Interrupt vector base address register.
MSR	Machine Status Register EE, SPE and PR bits.
PIR	Processor ID register.
SEMA4_Gatexx	Multicore: 1 SEMA4 gate is reserved for use by the OS. Default is Gate 0. It must be cache-inhibited and guarded. There is no reservation if the software semaphore option is used.
SPRG4-7/PMGC0	Multicore: Register used to store the cached core ID.

7.3 Required OS resources

RTA-OS needs the following resources for correct operation.

Resource	Description
Cross-core interrupt	In multi-core configurations, the OS will allocate one free interrupt per core to use for cross-core communication
IVOR8	This is the system trap. The OS needs to use this when the configuration contains untrusted OS Applications.
SEMA4_Gatexx	In multi-core configurations, 1 SEMA4 gate is reserved for use by the OS. Default is Gate 0. It must be cache-inhibited and guarded. There is no reservation if the software semaphore option is used.

7.3.1 Core ID Caching

In multi-core configurations it is necessary to cache the AUTOSAR core ID into a register that can be read efficiently by trusted and untrusted code. The target option 'Cached CoreID register' is used to specify which register to use from one of SPRG4, SPRG5, SPRG6, SPRG7 or PMGC0. The default is either SPRG4 or PMGC0 (variant dependent). The Os_CacheCoreID() must be called to do this caching for you. It must be called on each core before any other OS call is made (including GetCoreID()). If you use the OS_MAIN() macro, then this will silently call Os_CacheCoreID() for you. Similarly it will be called during the execution of Os_InitializeVectorTable().

7.4 Interrupts

This section explains the implementation of RTA-OS's interrupt model on the MPC5xxx/SPC5xx.

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 MPC5xxx/SPC5xx:

IPL	INTC_CPR	Description
0	EE=1, INTC_CPR=0	User (task) level
1-15/63	EE=1, INTC_CPR=1-15/63	Category 1 and 2 level (Max value depends upon target)
16/64	EE=0, INTC_CPR=x	Category 1 only (Max value depends upon target)

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

RTA-OS provides the APIs EnableInterruptSource(), DisableInterruptSource() and ClearPendingInterrupt(). However, the ClearPendingInterrupt() API is not supported on this target because it is not feasible to do so. Calling the API will result in E_OS_ID being returned. The ClearPending parameter in the EnableInterruptSource() API is ignored.

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 MPC5xxx/SPC5xx. A ✓ indicates that the mapping is permitted and a ✗ indicates that it is not permitted:

Address	Category 1	Category 2
INTC_0 to the highest INTC interrupt. Multicore operation uses software interrupts for cross-core communication (By default these are allocated from INTC_0).	✓	✓
IVOR_0 to the highest IVOR interrupt.	✓	✗

RTA-OS normally selects hardware vectoring mode when there are INTC interrupts and it will ensure that the INTC vector table is configured correctly. You can alternatively use a target option to select software vectoring mode, in which case RTA-OS will supply a dispatcher attached to IVOR_4 to perform dispatching of INTC based interrupts. It also performs SPE/EFPU2 preservation if the 'Preserve SPE' target option is enabled. The Os_INTC_vectors table changes in software vectoring mode to contain pointers to void functions that take a single uint32 argument containing the vector number 0, 1, 2 etc. If you place a Category 1 ISR on IVOR_4 (External Interrupt) then RTA-OS will not generate a dispatcher but instead allow you to write your own. In your dispatcher you can use the following. The function to call for a Category 1 ISR 'name' is 'void <name>(vector)'.

The function to call for a Category 2 ISR is void 'Os_ISRWrapper(vector)' for single core, 'Os_ISRWrapper<corenum>(vector)' for multi core.

Note that your CAT1_ISR handler code is entered with all necessary CPU registers saved, but no others.

With software vectoring, typical Category 1 ISRs should not signal end of interrupt by writing to the EOIR register in the interrupt controller. This is because the EOIR is done in the software dispatcher. The exception to this is where the Category 1 ISR does not return normally - as in the case where Os_TimingFaultDetected is called from a Category 1 Timing Protection interrupt, which results in termination of the overrunning code. In this case EOIR will need to be implemented in the ISR itself, before calling Os_TimingFaultDetected.

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
Os_intvec	Contains the INTC vectors. The linker/locater must ensure that alignment is appropriate for the target variant.
Os_cpuvec	Contains the CPU vectors/initializers. The linker/locater must ensure that alignment is appropriate for the target variant.

When using hardware vectoring on variants where the Os_cpuvec section must be located first, then there is typically an offset which must be adhered to between it and the placing of the Os_intvec section. This offset can vary depending upon the variant and is based on the size of the unique hardwired vector number in the INTC_IACKR register, plus 2 bits to allow for a resultant 4 byte aligned vector address, when combined with the IVPR register. For example, on the SPC58EC80 the size of the hardwired vector number is 10 bits, plus the 2 bits to give a 4 byte alignment results in 12 bits in total and that equates to an offset of 4096 or 0x1000. The Os_intvec section should be located 4096 after the Os_cpuvec section. Refer to the 'Interrupt Controller (INTC)' section in the selected variants reference manual for more details.

With hardware vectoring, Category 1 ISRs should signal end of interrupt by writing to the EOIR register in the interrupt controller. If your interrupt used the CAT1_ISR macro, then this can be done automatically for you by setting the target option 'Generate Cat1 EOIR' to TRUE.

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.

RTA-OS reserves the IVOR_8 (System Call) vector for applications that use untrusted code to allow switching between trusted (supervisor level) and untrusted (user level) code. This functionality must still be supported if a user provided handler is used in such applications.

When 'Suppress Vector Table Generation' is configured as TRUE, no CPU or INTC vector tables will be generated. You are then responsible for providing them. You should still call Os_InitializeVectorTable() to ensure that interrupt priorities are correctly configured. Where there are multiple cores, each core must call Os_InitializeVectorTable().

7.4.4 Using Raw Exception Handlers

RTA-OS supports direct branches in the interrupt vector table for Category 1 interrupts placed on the IVOR vectors. Normally RTA-OS produces wrapper code around the interrupt handler functions for these exceptions to ensure that the necessary context is preserved. If interrupt handlers are given names starting with 'b_' then the interrupt vector table entry is an unconditional branch 'e_b' instruction to the handler function. When using these raw exception handlers it is the user's responsibility that:

- The correct register context is saved and restored.
- The correct return instruction is used.
- Interrupts are not re-enabled in these handlers.
- The RTA-OS API is not used in these handlers.

7.4.5 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 */
}
```

7.4.6 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.

7.4.7 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 7.4.5 for further details).

7.5 Memory Model

The following memory models are supported:

Model	Description
Flat 32 bit address space	The SDA small data/const area thresholds default to zero, preventing small data/const access. These can be changed via target options.

7.6 Processor Modes

RTA-OS can run in the following processor modes:

Mode	Notes
Supervisor	All OS and "trusted" code runs in supervisor mode. Note that if "trusted-with-protection" code is used, the Os_Cbk_GetSetProtection() callback must be implemented. (Documented in the RTA-OS Reference Guide.)
User	All "untrusted" code runs in user mode.

7.7 Stack Handling

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

RTA-OS uses the stack in use when the OS starts (register R1). Where there are multiple cores, each core must use different stack areas.

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 PPCe200/WR 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	MPC5746Mv2 on MPC57XX EVB
CPU Clock Speed	16.0MHz
Stopwatch Speed	16.0MHz

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:

- `Os_Cfg_Counters` includes data for counters, alarms and schedule tables.
- `Os_Cfg` contains the data for most other OS objects.

8.2.1 Single Core

The following table gives the ROM and/or RAM requirements (in bytes) for each OS object in a simple single-core 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	16
Task	20	0

8.2.2 Multi Core

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

Object	ROM	RAM
Alarm	8	12
Cat 2 ISR	16	0
Core Overheads (each OS core)	0	60
Core Overheads (each processor core)	32	28
Counter	32	4
CounterCallback	4	0
ExpiryPoint	3.5	0
OS Overheads (max)	0	9
OS-Application	8	0
PeripheralArea	0	0
Resource	16	4
ScheduleTable	20	16
Task	36	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

8.4.1 Single Core

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 single-core configuration in standard status.

Library Module	.Os_text_vle	.sbss	.text_vle	.wrc.metadata	COMMON	Os_cpuvec	Os_intvec
ActivateTask			176				
AdvanceCounter			4				
CallTrustedFunction			34				
CancelAlarm			152				
ChainTask			158				
CheckISRMemoryAccess			64				
CheckObjectAccess			144				
CheckObjectOwnership			144				
CheckTaskMemoryAccess			64				
ClearEvent			80				
ControlIdle			96	4			
DisableAllInterrupts		8	62				
DispatchTask			196				
ElapsedTime			408				
EnableAllInterrupts			70				
GetActiveApplicationMode			6				
GetAlarm			202				
GetAlarmBase			52				
GetApplicationID			32				
GetCounterValue			86				
GetCurrentApplicationID			32				
GetElapsedCounterValue			116				
GetEvent			80				
GetExecutionTime			80				
GetISRID			6				
GetIsrMaxExecutionTime			80				

Library Module	.Os_text_vle	.sbss	.text_vle	.wrc.metadata	COMMON	Os_cpuvec	Os_intvec
GetIsrMaxStackUsage			80				
GetResource			108				
GetScheduleTableStatus			86				
GetStackSize			4				
GetStackUsage			80				
GetStackValue			10				
GetTaskID			10				
GetTaskMaxExecutionTime			80				
GetTaskMaxStackUsage			80				
GetTaskState			34				
GetVersionInfo			20				
Idle			4				
InShutdown			2				
IncrementCounter			26				
InterruptSource			314				
ModifyPeripheral			300				
MultiCoreInit	114						
NextScheduleTable			172				
Os_Cfg		33	1136		592		
Os_Cfg_Counters			6178				
Os_Cfg_KL			46				
Os_GetCurrentIMask			22				
Os_GetCurrentTPL			26				
Os_StartCores			104				
Os_SysCall	4						
Os_Vectors						376	38
Os_Wrapper			158				
Os_setjmp	194						
Os_vec_init			274				
ProtectionSupport			106				
ReadPeripheral			252				
ReleaseResource			146				
ResetIsrMaxExecutionTime			80				
ResetIsrMaxStackUsage			80				
ResetTaskMaxExecutionTime			80				
ResetTaskMaxStackUsage			80				
ResumeAllInterrupts			70				
ResumeOSInterrupts			70				
Schedule			162				
SetAbsAlarm			156				

Library Module	.Os_text_vle	.sbss	.text_vle	.wrc.metadata	COMMON	Os_cpuvec	Os_intvec
SetEvent			80				
SetRelAlarm			212				
SetScheduleTableAsync			118				
ShutdownOS			68				
StackOverrunHook			14				
StartOS			140				
StartScheduleTableAbs			170				
StartScheduleTableRel			162				
StartScheduleTableSynchron			118				
StopScheduleTable			136				
SuspendAllInterrupts		8	62				
SuspendOSInterrupts		8	110				
SyncScheduleTable			118				
SyncScheduleTableRel			118				
TerminateTask			16				
ValidateCounter			56				
ValidateISR			20				
ValidateResource			38				
ValidateScheduleTable			38				
ValidateTask			40				
WaitEvent			80				
WritePeripheral			246				

8.4.2 Multi Core

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 multi-core configuration in standard status.

Library Module	.Os_text_vle	.sbss	.text_vle	.wrc.metadata	COMMON	Os_cpuvec	Os_intvec
ActivateTask			324				
AdvanceCounter			4				
CallTrustedFunction			34				
CancelAlarm			198				
ChainTask			226				

Library Module	.Os_text_vle	.sbss	.text_vle	.wrc.metadata	COMMON	Os_cpuvec	Os_intvec
CheckISRMemoryAccess			64				
CheckObjectAccess			236				
CheckObjectOwnership			162				
CheckTaskMemoryAccess			64				
ClearEvent			96				
ControlIdle			122	8			
CrossCore			52				
DisableAllInterrupts			68				
DispatchTask			446				
ElapsedTime			488				
EnableAllInterrupts			76				
GetActiveApplicationMode			6				
GetAlarm			214				
GetAlarmBase			48				
GetApplicationID			56				
GetCounterValue			102				
GetCurrentApplicationID			56				
GetElapsedCounterValue			136				
GetEvent			96				
GetExecutionTime			96				
GetISRID			22				
GetIsrMaxExecutionTime			96				
GetIsrMaxStackUsage			96				
GetNumberOfActivatedCores			30				
GetResource			122				
GetScheduleTableStatus			152				
GetSpinlock			4				
GetStackSize			4				
GetStackUsage			96				
GetStackValue			26				
GetTaskID			26				
GetTaskMaxExecutionTime			96				
GetTaskMaxStackUsage			96				
GetTaskState			96				
GetVersionInfo			20				
Idle			4				
InShutdown			2				
IncrementCounter			26				
InterruptSource			364				
ModifyPeripheral			348				

Library Module	.Os_text_vle	.sbss	.text_vle	.wrc.metadata	COMMON	Os_cpuvec	Os_intvec
MultiCoreInit	114						
NextScheduleTable			222				
Os_CacheCoreID			14				
Os_Cfg		17	2046		744		
Os_Cfg_Counters			7624				
Os_Cfg_KL			110				
Os_CoreLocks			96				
Os_CrossCore			348				
Os_GetCurrentIMask			36				
Os_GetCurrentTPL			96				
Os_ScheduleQ			76				
Os_StartCores			104				
Os_SysCall	4						
Os_Vectors						376	38
Os_Wrapper			180				
Os_setjmp	194						
Os_vec_init			318				
ProtectionSupport			122				
ReadPeripheral			300				
ReleaseResource			160				
ReleaseSpinlock			4				
ResetIsrMaxExecutionTime			96				
ResetIsrMaxStackUsage			96				
ResetTaskMaxExecutionTime			96				
ResetTaskMaxStackUsage			96				
ResumeAllInterrupts			76				
ResumeOSInterrupts			76				
Schedule			168				
SetAbsAlarm			206				
SetEvent			96				
SetRelAlarm			272				
SetScheduleTableAsync			134				
ShutdownAllCores			98				
ShutdownOS			106				
StackOverrunHook			14				
StartCore			64				
StartNonAutosarCore			64				
StartOS			340				
StartScheduleTableAbs			218				
StartScheduleTableRel			208				

Library Module	.Os_text_vle	.sbss	.text_vle	.wrc.metadata	COMMON	Os_cpuvec	Os_intvec
StartScheduleTableSynchron			134				
StopScheduleTable			182				
SuspendAllInterrupts			68				
SuspendOSInterrupts			124				
SyncScheduleTable			134				
SyncScheduleTableRel			134				
TerminateTask			30				
TryToGetSpinlock			8				
ValidateCounter			38				
ValidateISR			20				
ValidateResource			38				
ValidateScheduleTable			56				
ValidateTask			72				
WaitEvent			96				
WritePeripheral			294				

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:

Time in microseconds = Time in cycles / CPU Clock rate in MHz

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

- at 20MHz = 50/20 = 2.5μs
- at 80MHz = 50/80 = 0.625μs
- at 150MHz = 50/150 = 0.333μs

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).

Values are given for single-core operation only. Timings for cross-core activations, though interesting, are variable because of the nature of multi-core operation. Minimum values cannot be given, because timings are dependent on the activity on the core that receives the activation.

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()`).

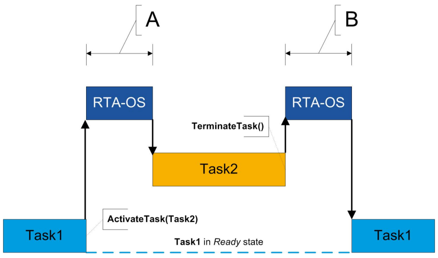
Interrupt latency is the time between an interrupt request being recognized by the target hardware and the execution of the first instruction of the user provided handler function:

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

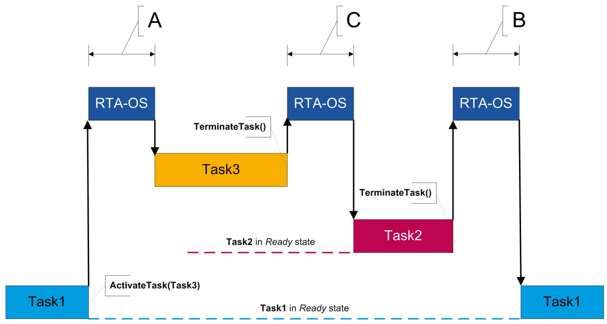
For Category 2 ISRs this is the time required for the hardware to recognize the interrupt plus the time required by RTA-OS to set-up the context in which the ISR runs.

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

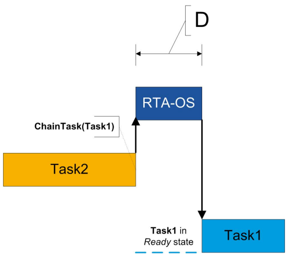
Switch	Key	CPU Cycles	Actual Time
Task activation	A	184	11.5us
Task termination with resume	B	112	7us
Task termination with switch to new task	C	133	8.31us
Chaining a task	D	246	15.4us
Waiting for an event resulting in transition to the WAITING state	E	568	35.5us
Setting an event results in task switch	F	737	46.1us
Non-preemptive task offers a preemption point (co-operative scheduling)	G	189	11.8us
Releasing a resource results in a task switch	H	179	11.2us
Entering a Category 2 ISR	I	82	5.12us
Exiting a Category 2 ISR and resuming the interrupted task	J	169	10.6us
Exiting a Category 2 ISR and switching to a new task	K	209	13.1us
Entering a Category 1 ISR	L	29	1.81us



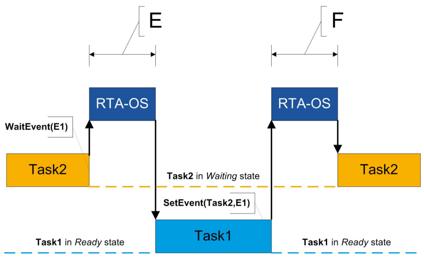
(a) Task activated. Termination resumes preempted task.



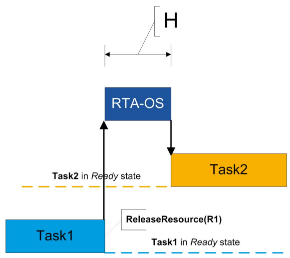
(b) Task activated. Termination switches into new task.



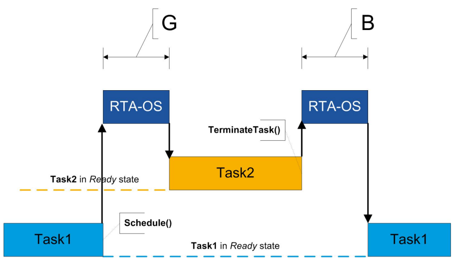
(c) Task chained.



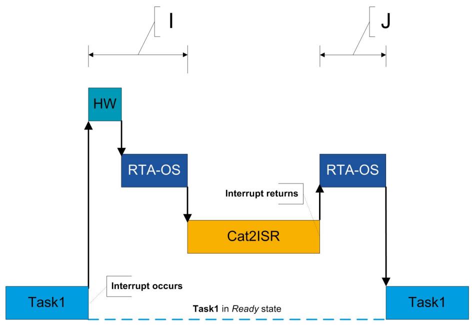
(d) Task waits. Task is resumed when event set.



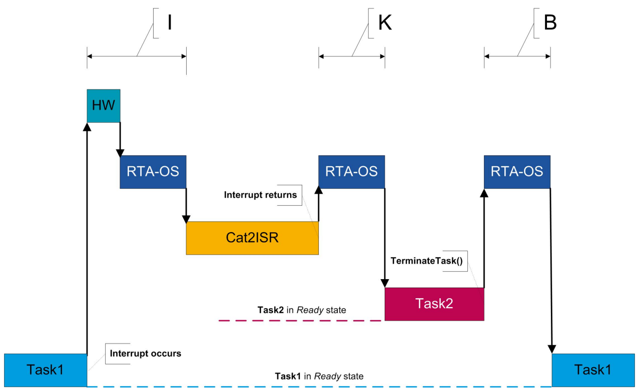
(e) Task switch when resource is released.



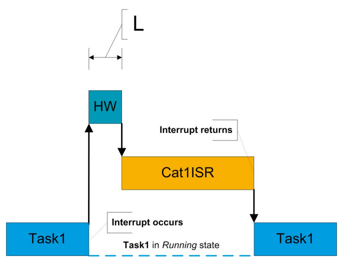
(f) Request for scheduling made by non-preemptive task.



(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 PPCe200/WR-specific parts of RTA-OS can be found in the following manuals:

***PPCe200/WR Release Note*.** This document provides information about the PPCe200/WR 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 Section 10.2.2).

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.uk@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
- Your `.xml`, `.arxml`, `.rtaos` and/or `.stc` files
- The command line which caused the error
- The version of the ETAS tools you are using
- The version of the compiler tool chain you are using
- The error message you received (if any)
- The file `Diagnostic.dmp` if it was generated

10.2 General Enquiries

10.2.1 ETAS Global Headquarters

ETAS GmbH		
Borsigstrasse 24	Phone:	+49 711 3423-0
70469 Stuttgart	Fax:	+49 711 3423-2106
Germany	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, 38
AUTOSAR OS includes
- Os.h, 29
- Os_Cfg.h, 29
- Os_MemMap.h, 29

C

CAT1_ISR, 33
Compiler, 37
Compiler (WindRiver (Diab) v5.8.0.0 patch 5 (MPC5676R/MPC5777C only)), 36
Compiler (WindRiver (Diab) v5.8.0.0 patch 6 (MPC5676R/MPC5777C only)), 36
Compiler (WindRiver (Diab) v5.9.4.0), 36
Compiler (WindRiver (Diab) v5.9.4.7), 37
Compiler (WindRiver (Diab) v5.9.6.1), 37
Compiler (WindRiver (Diab) v5.9.6.2), 37
Compiler (WindRiver (Diab) v5.9.6.6), 37
Compiler Versions, 36
Configuration
- Port-Specific Parameters, 21

Core ID Caching, 46

D

Debugger, 40

E

ETAS License Manager, 11
- Installation, 11

F

Files, 29

H

Hardware
- Requirements, 9

I

Installation, 9
- Default Directory, 10
- Verification, 19

Interrupts, 46
- Category 1, 49
- Category 2, 49
- Default, 50

IPL, 47

L

Librarian, 39
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, 39

M

Memory Model, 50

O

Options, 37
Os_CacheCoreID, 30
Os_Cbk_GetAbortStack, 31
Os_Cbk_StartCore, 32
Os_Disable_x, 34
Os_DisableAllConfiguredInterrupts, 33
Os_Enable_x, 34
Os_EnableAllConfiguredInterrupts, 34
Os_InitializeVectorTable, 31
Os_IntChannel_x, 34
Os_StackSizeType, 35
Os_StackValueType, 35

P

Parameters of Implementation, 21
Performance, 51
- Context Switching Times, 59
- Library Module Sizes, 53
- RAM and ROM, 51
- Stack Usage, 52

Processor Modes, 50
- Supervisor, 50
- User, 50

R

Registers
- CCU, 45
- Initialization, 45
- INTC_BCR, 46
- INTC_CPR, 46
- INTC_EOIR, 46
- INTC_IACKR, 46
- INTC_PSRx, 45, 46
- IVORx, 45
- IVPR, 45, 46
- L1CSR, 45
- MSR, 45, 46
- Non-modifiable, 46
- PIR, 46
- SEMA4_Gatexx, 46
- SPRG4-7/PMGC0, 45, 46
- TLB, 45
- XBAR, 45

Resource
- Cross-core interrupt, 46
- IVOR8, 46
- SEMA4_Gatexx, 46

S

Software
- Requirements, 9

Stack, 50

T

Target, 42
- Variants, 45

Toolchain, 36

U

Using Raw Exception Handlers, 49

V

Variants, 45

Vector Table
- Base Address, 48