RTA-OS

OMAP35x/RVDS Port Guide

Copyright

Document: 10481-PG-2.0.1

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

RTA-OS is a small and fast real-time operating system that conforms to both the AUTOSAR OS (R3.0.0, R3.1.2 to R4.0.0, MultiCore) and OSEK/VDX 2.2.3 standards. The operating system is configured and built on a PC, but runs on your target hardware.

This document describes the RTA-OS OMAP35x/RVDS port plug-in that customizes the RTA-OS development tools for the Texas Instruments OMAP35x/AM35x with the ARM RVDS 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 OMAP35x/RVDS port and cover:

- how to install the OMAP35x/RVDS port plug-in;
- how to configure OMAP35x/RVDS-specific attributes;
- how to build an example application to check that the OMAP35x/RVDS 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 OMAP35x/AM35x, 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 are printed in **bold, blue** characters.
Click **OK**.	Button labels are printed in **bold** characters
Press <Enter>.	Key commands are enclosed in angle brackets.
The “Open file” dialog box appears	The names of program windows, dialog boxes, 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.	Hyperlinks through the document are shown in blue letters.
	Functionality that is provided in RTA-OS but may not be portable to another AUTOSAR OS implementation is marked with the ETAS logo.

Caution! Notes like this contain important instructions that you must follow carefully in order for things to work correctly.

1.3 References

OSEK is a European automotive industry standards effort to produce open systems interfaces for vehicle electronics. 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.

You must have installed the RTA-OS tools before installing the OMAP35x/RVDS 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 (or higher) IBM compatible PC.
- 512Mb RAM.
- 500Mb 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 2000 (Service Pack 3 or later)
- Windows XP (Service Pack 2 or later)
- Windows Vista
- Windows 7

The tools provided with RTA-OS require Microsoft's .NET Framework v2.0 *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* `http://www.microsoft.com/net/Download.aspx`*.*

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 OMAP35x/RVDS 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 OMAP35x/RVDS 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.

*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 `OMAP35xRVDS_2.0.1` . This contains everything to do with the port plug-in.

Each version of the port installs in its own directory - the trailing `_2.0.1` 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:

OMAP35xRVDS.dll - the port plug-in that is used by **rtaosgen** and **rtaoscfg**.

RTA-OS OMAP35xRVDS Port Guide.pdf - the documentation for the port (the document you are reading now).

RTA-OS OMAP35xRVDS 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 issues.

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.

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

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;

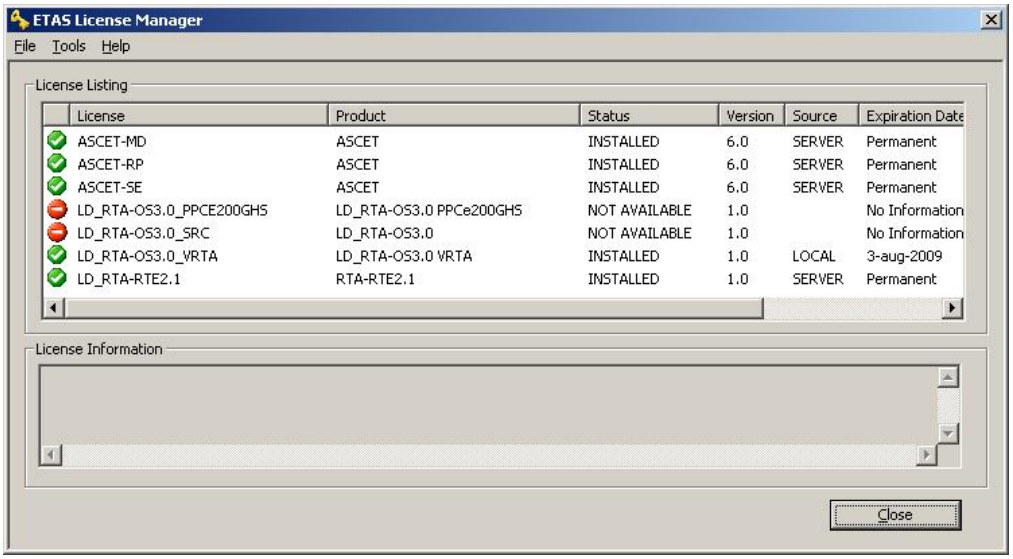


Figure 2.1: The ETAS License manager

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 seven 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.

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
```

Figure 2.2: Obtaining License Information

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.

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 RTA-OS without a valid license, you will be given the opportunity to start the ETAS License Manager and select a license.

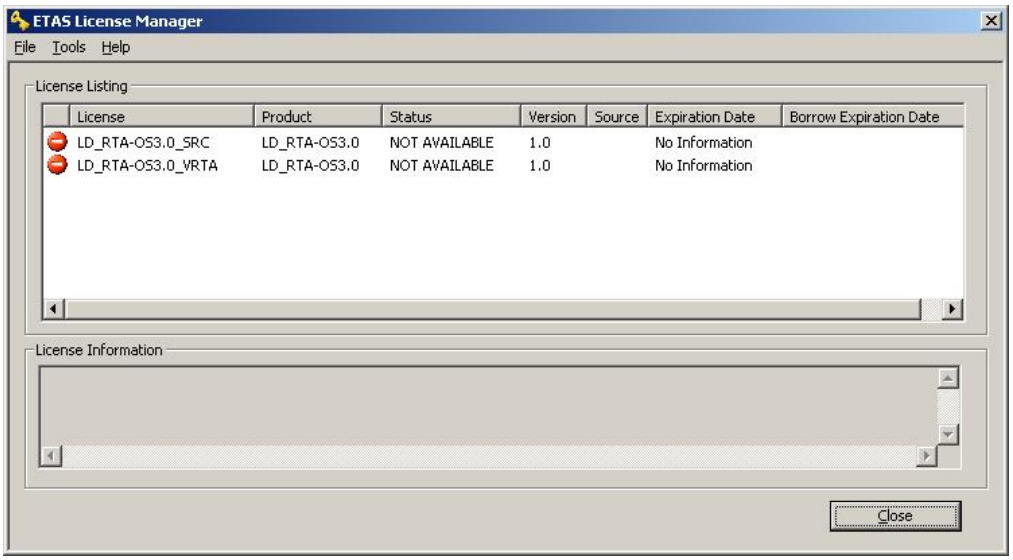


Figure 2.3: Unlicensed RTA-OS Installation

When the ETAS License Manager is launched, it will display the RTA-OS license state as `NOT AVAILABLE` and you will not be able to use any of the tools until a valid license is installed. This is shown in Figure 2.3.

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:

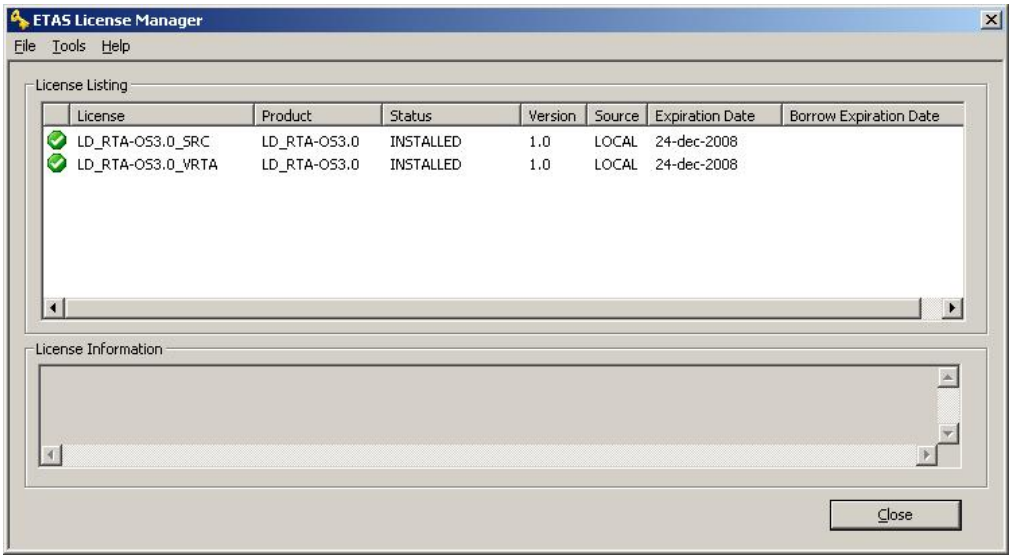


Figure 2.4: Licensed features for RTA-OS

```
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.

When a license is installed by the ETAS License Manager it is placed in: `C:\Documents and Settings\All Users\Application Data\ETAS\FlexNet`

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”

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`.

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.

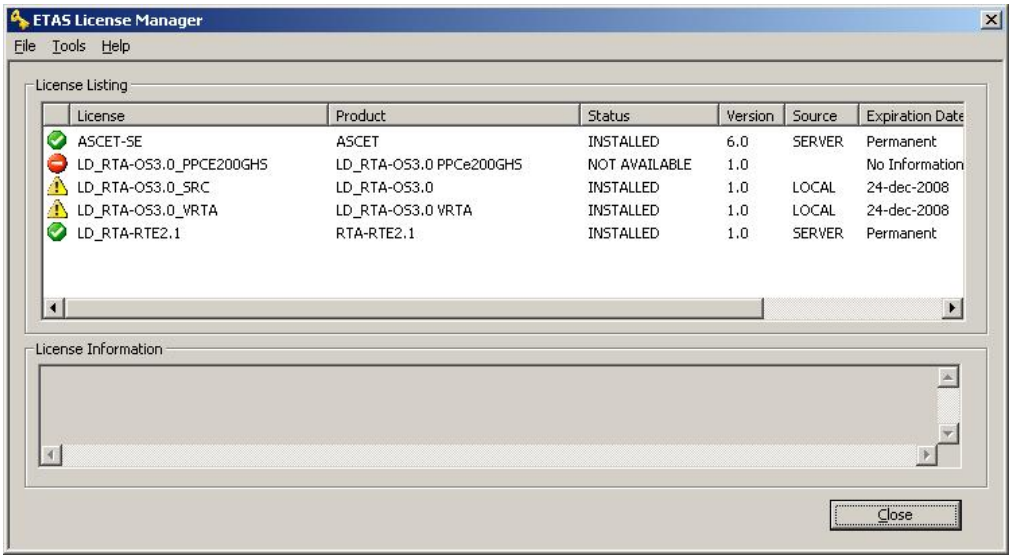


Figure 2.5: Licensed features that are due to expire

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 → About** or **Help → Version Information** drop down menus. These will show information about your complete RTA-OS configuration.

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]
```

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.

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 OMAP35x/RVDS port.

4.1 Parameters of Implementation

To be a valid OSEK 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 OMAP35x/RVDS 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 OMAP35x/RVDS 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 Os_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. The value does not normally change if the OS configuration changes.

4.2.3 Stack overheads for ISR activation

XML name SpIDisp

Description

The 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 the OS may need to use marginally more stack than if it activates a task that is of lower priority. 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 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, assuming the maximum depth of preemption that can occur. 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

Select ORTI generation for the Lauterbach debugger.

Settings

Value	Description
TRUE	Generate ORTI
FALSE	No ORTI

4.2.7 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)

4.2.8 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

24 Port Characteristics

Settings

Value	Description
TRUE	Perform the checks (default)
FALSE	Do not check

4.2.9 Compiler Version

XML name CompilerVersion

Description

Mandate the version of the ARM RVDS Compiler to use.

Settings

Value	Description
Off	Use any supported compiler
v4.0	Test that the 'v4.0' compiler is used
v4.1	Test that the 'v4.1' compiler is used

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.
`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 OMAP35x/RVDS 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_InitializeVectorTable

Initialize the INTC_MIRx, INTCPS_IPLm and CPSR registers. Moves the CPSR to SYS mode and transfers the current stack pointer value to the SYS/USR mode stack

Syntax

```
void Os_InitializeVectorTable(void)
```

Description

Os_InitializeVectorTable() initializes the INTC_MIRx, INTCPS_IPLm and CPSR according to the requirements of the project configuration.

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

6 Toolchain

This chapter contains important details about RTA-OS and the ARM RVDS 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.

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 ARM RVDS v4.0 [Build 821] and v4.1 [Build 561] compilation tools. The v4.0 compiler uses the -Otime command line option in addition to those used with the v4.1 compiler detailed in section 6.1. When building the RTA-OS library the –no_multifile command line option is used with the v4.0 compiler. All performance figures in this manual have been measured using the v4.1 tools.

6.1 Compiler

Name armcc.exe
Version ARM C/C++ Compiler, 4.1 [Build 713]

Options

Kernel Options

The following options are used to build the RTA-OS kernel:

`-thumb` Use the Thumb2 instruction set

`-cpu=Cortex-A8` Generate code for the Cortex-A8 CPU

`-fpu=none` Do not generate floating point code

`-O3` Set maximum level of optimization

`-Otime` Perform optimizations to reduce execution time

`-autoinline` Enable automatic inlining of functions

`-diag_style=gnu` Diagnostic messages are in the gnu style

Mandatory Options for Application Code

The following options are mandatory for application code (for this configuration):

- The same options as for compilation

Forbidden Options for Application Code

The following options are forbidden for application code (for this configuration):

`-bigend` Generate code for big-endian memory addressing

6.2 Assembler

Name armcc.exe
Version ARM C/C++ Compiler, 4.1 [Build 713]

Options

Kernel Options

The following options are used to build the RTA-OS kernel:

- The same options as for compilation

Mandatory Options for Application Code

The following options are mandatory for application code (for this configuration):

- The same options as for compilation

Forbidden Options for Application Code

The following options are forbidden for application code (for this configuration):

- Any options that conflict with kernel options

6.3 Librarian

Name armar.exe
Version ARM Librarian, 4.1 [Build 713]

6.4 Linker

Name armlink.exe
Version ARM Linker, 4.1 [Build 713]

Options

Kernel Options

The following options are used to build the RTA-OS kernel:

`-info totals,sizes,unused` Specify map file contents

`-datacompressor=off` Disable RW data compression

`-no_remove` Do not remove unused input sections

`-xref` Output cross reference information to the map file

`-map` Output memory map to the map file

`-symbols` Output symbol table to the map file

`-verbose` Output detailed information to the map file

`-entry=reset_handler` Specify the application entry point

Mandatory Options for Application Code

The following options are mandatory for application code (for this configuration):

- The same options as for compilation

Forbidden Options for Application Code

The following options are forbidden for application code (for this configuration):

-be8 Produce little-endian code and big-endian data

-be32 Produce big-endian code and big-endian data

6.5 Debugger

Name Lauterbach TRACE32
Version Build 10654 or later

7 Hardware

7.1 Supported Devices

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

Name: Texas Instruments
Device: OMAP35x/AM35x

The following variants of the OMAP35x/AM35x are supported:

- AM3505
- AM3517
- Generic35x
- OMAP3503
- OMAP3515
- OMAP3525
- OMAP3530

If you require support for a variant of OMAP35x/AM35x 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
CPSR	The CPSR must select a privileged mode (i.e. SVC or SYS) before calling Os_InitializeVectorTable().
INTC_ILRx	The INTC priorities have to be set to the values declared in the configuration. This can be done by calling Os_InitializeVectorTable().
INTC_MIRx	The INTC mask registers have to be set to match the declared ISRs in the configuration. This can be done by calling Os_InitializeVectorTable().
SP	The stack must be allocated and SP initialized before calling Os_InitializeVectorTable().

7.2.2 Modification

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

Register	Notes
CPSR	User code may not change the operating mode.
INTC	User code may not program the INTC directly.
SP	User code may not change the USR/SYS stack pointer other than as a result of normal program flow.

RTA-OS operates all code with the CPSR.A bit enabled.

7.3 Interrupts

This section explains the implementation of RTA-OS's interrupt model on the OMAP35x/AM35x.

7.3.1 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 OMAP35x/AM35x:

IPL	INTCPS_THRESHOLD	Description
0	63	User (task) level. No interrupts are masked.
1	62	Maskable Category 1 and 2 interrupts routed through IRQ.
...	...	Maskable Category 1 and 2 interrupts routed through IRQ.
61	2	Maskable Category 1 and 2 interrupts routed through IRQ.
62	1	Maskable Category 1 interrupts routed through FIQ.
63	0	Spurious (FIQ or IRQ) INTC interrupt handler
64	n/a	Non INTC Category 1 exceptions.

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

7.3.2 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 OMAP35x/AM35x. A ✓ indicates that the mapping is permitted and a ✗ indicates that it is not permitted:

Address	Category 1	Category 2
0x4, 0x8, 0xC, 0x10 exception handlers	✓	✗
0x20 Spurious (FIQ or IRQ) INTC interrupt handler	✓	✗
0x24 to 0x1A0 INTC interrupt handlers	✓	✓

RTA-OS requires that on entry to IRQ and FIQ exceptions the Cortex CPU switchs to the ARM instruction set. RTA-OS will the switch to the Thumb instruction set before entering application interrupt handler code. The Cortex CPU instruction set state must not be modified from the default setting and make exceptions use the Thumb instruction set on exception entry.

7.3.3 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_ExceptionVectors	Should either be located at absolute address 0x4020FFC8 where the boot ROM routes all CPU exceptions, or located in accordance with the Cortex Vector Base Address Register. The first entry is the undefined instruction handler.

The RTA-OS generated vector table does not include the reset vector. This should be added for an application and located before the undefined instruction handler.

7.3.4 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 */
}
```

Category 1 Interrupt handlers routed to the Cortex CPU exceptions must not use the CAT1_ISR macro and must be decorated with the __irq modifier (i.e. Undefined instruction, SVC, Prefetch abort and Data abort).

7.3.5 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.

7.3.6 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 7.3.4 for further details).

7.4 Memory Model

The following memory models are supported:

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

Apart from some small code sections RTA-OS uses the default compiler memory sections unless modified by the AUTOSAR memmap.h overrides. The non-default code sections all use the prefix 'Os_' (i.e. Os_primitives)

7.5 Processor Modes

RTA-OS can run in the following processor modes:

Mode	Notes
Trusted	All trusted code runs in system (SYS) mode.
Untrusted	All untrusted code runs in user (USR) mode.

RTA-OS uses the SVC handler to transfer between Untrusted and Trusted code in Autosar SC3/SC4 applications. This functionality must be supported if a user provided SVC handler is used in such applications.

7.6 Stack Handling

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

RTA-OS manages the USR/SYS stack (via register SP). No IRQ or FIQ stack is used. The RTA-OS function Os_InitializeVectorTable() transfers the current stack pointer value to the SYS/USR stack.

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 OMAP35x/RVDS 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	OMAP3530 on Beagleboard Rev C3
CPU Clock Speed	26.0MHz
Stopwatch Speed	13.0MHz

8.2 Memory Consumption

8.2.1 OS Object RAM and ROM Usage

Each OS object requires ROM and/or RAM. The following table gives the ROM and/or RAM requirements (in bytes) for each OS object in the RTA-OS Component. 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
Resource	8	4
ScheduleTable	16	16
Task	16	0

The OS objects, and any object-specific code required to manipulate them, are generated by **rtaosgen** and placed in the RTA-OS library:

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

8.2.2 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.

8.2.3 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 RTA-OS in standard status.

Library Module	Code	RO Data	RW Data	ZI Data
ActivateTask.o	100	0	0	0

Library Module	Code	RO Data	RW Data	ZI Data
AdvanceCounter.o	4	0	0	0
CallTrustedFunction.o	24	0	0	0
CancelAlarm.o	84	0	0	0
ChainTask.o	100	0	0	0
CheckISRMemoryAccess.o	36	0	0	0
CheckObjectAccess.o	4	0	0	0
CheckObjectOwnership.o	4	0	0	0
CheckTaskMemoryAccess.o	36	0	0	0
ClearEvent.o	36	0	0	0
DisableAllInterrupts.o	56	0	8	0
DispatchTask.o	152	0	0	0
EnableAllInterrupts.o	40	0	0	0
GetActiveApplicationMode.o	12	0	0	0
GetAlarm.o	144	0	0	0
GetAlarmBase.o	44	0	0	0
GetApplicationID.o	4	0	0	0
GetCounterValue.o	52	0	0	0
GetElapsedCounterValue.o	76	0	0	0
GetEvent.o	36	0	0	0
GetExecutionTime.o	36	0	0	0
GetISRID.o	12	0	0	0
GetIsrMaxExecutionTime.o	36	0	0	0
GetIsrMaxStackUsage.o	36	0	0	0
GetResource.o	80	0	0	0
GetScheduleTableStatus.o	52	0	0	0
GetStackSize.o	4	0	0	0
GetStackUsage.o	36	0	0	0
GetStackValue.o	16	0	0	0
GetTaskID.o	16	0	0	0
GetTaskMaxExecutionTime.o	36	0	0	0
GetTaskMaxStackUsage.o	36	0	0	0
GetTaskState.o	40	0	0	0
GetVersionInfo.o	36	0	0	0
Idle.o	4	0	0	0
InShutdown.o	2	0	0	0
IncrementCounter.o	10	0	0	0
NextScheduleTable.o	116	0	0	0
Os_Cfg.o	200	648	28	592
Os_Cfg_Counters.o	4192	728	0	0
Os_ExceptionVectors.o	188	0	0	0

Library Module	Code	RO Data	RW Data	ZI Data
Os_FIQHandler.o	108	0	0	0
Os_IRQHandler.o	152	0	0	0
Os_Stack.o	4	0	0	0
Os_Trust.o	8	0	0	0
Os_Wrapper.o	76	0	0	0
Os_primitives.o	28	0	0	0
Os_setjmp.o	28	0	0	0
Os_vec_init.o	88	185	0	0
ProtectionSupport.o	40	0	0	0
ReleaseResource.o	68	0	0	0
ResetIsrMaxExecutionTime.o	36	0	0	0
ResetIsrMaxStackUsage.o	36	0	0	0
ResetTaskMaxExecutionTime.o	36	0	0	0
ResetTaskMaxStackUsage.o	36	0	0	0
ResumeAllInterrupts.o	40	0	0	0
ResumeOSInterrupts.o	40	0	0	0
Schedule.o	80	0	0	0
SetAbsAlarm.o	92	0	0	0
SetEvent.o	36	0	0	0
SetRelAlarm.o	148	0	0	0
SetScheduleTableAsync.o	52	0	0	0
ShutdownOS.o	52	0	0	0
StackOverrunHook.o	6	0	0	0
StartOS.o	92	0	4	0
StartScheduleTableAbs.o	120	0	0	0
StartScheduleTableRel.o	104	0	0	0
StartScheduleTableSynchron.o	52	0	0	0
StopScheduleTable.o	72	0	0	0
SuspendAllInterrupts.o	56	0	8	0
SuspendOSInterrupts.o	60	0	8	0
SyncScheduleTable.o	52	0	0	0
TerminateTask.o	4	0	0	0
ValidateCounter.o	52	0	0	0
ValidateISR.o	16	0	0	0
ValidateResource.o	36	0	0	0
ValidateScheduleTable.o	36	0	0	0
ValidateTask.o	36	0	0	0
WaitEvent.o	36	0	0	0

8.3 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 apply irrespective 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 8.3.1 have an uncertainty of 1 CPU cycle(s).

8.3.1 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()`).

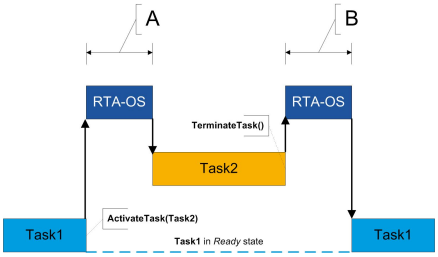
Interrupt latency is the time between an interrupt request being recognized by the target hardware and the execution of the first instruction of the user provided handler function:

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

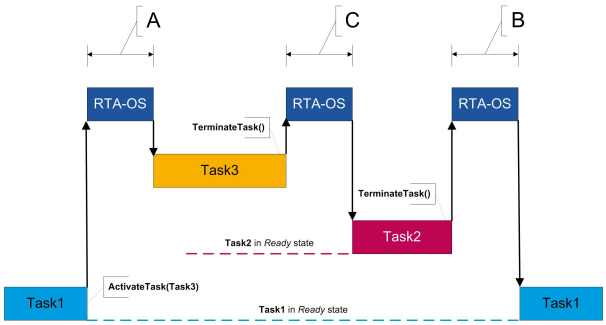
For Category 2 ISRs this is the time required for the hardware to recognize the interrupt plus the time required by RTA-OS to set-up the context in which the ISR runs.

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

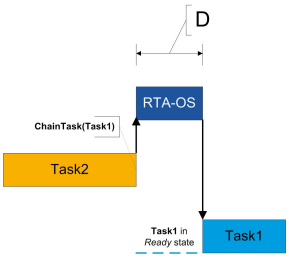
Switch	Key	Execution Time (CPU Cycles)
Task activation	A	210
Task termination with resume	B	110
Task termination with switch to new task	C	104
Chaining a task	D	228
Waiting for an event resulting in transition to the WAITING state	E	500
Setting an event results in task switch	F	726
Non-preemptive task offers a preemption point (co-operative scheduling)	G	214
Releasing a resource results in a task switch	H	204
Entering a Category 2 ISR	I	188
Exiting a Category 2 ISR and resuming the interrupted task	J	186
Exiting a Category 2 ISR and switching to a new task	K	218
Entering a Category 1 ISR	L	126



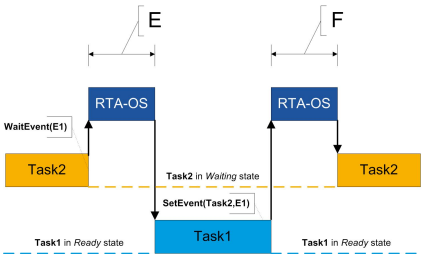
(a) Task activated. Termination resumes preempted task.



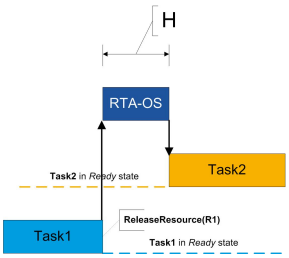
(b) Task activated. Termination switches into new task.



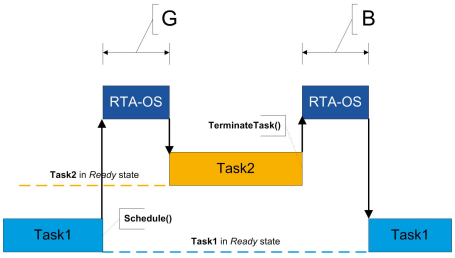
(c) Task chained.



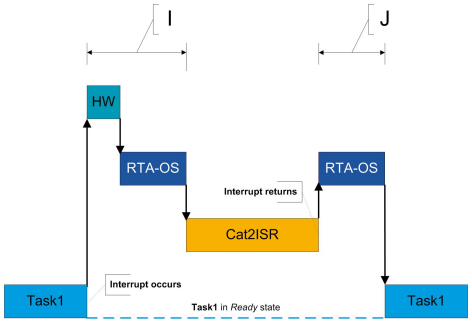
(d) Task waits. Task is resumed when event set.



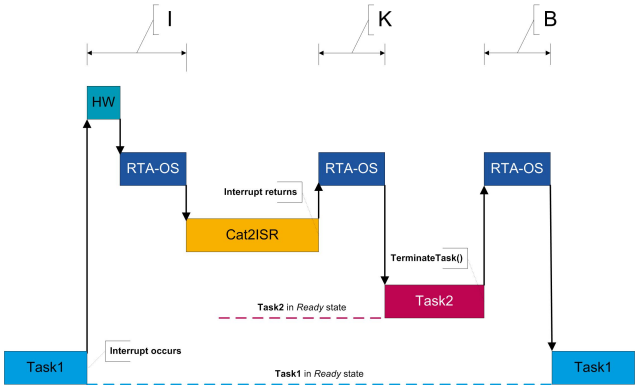
(e) Task switch when resource is released.



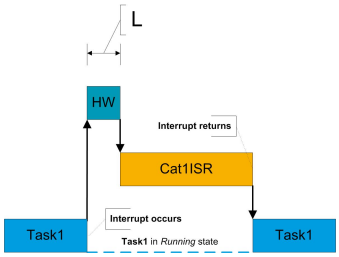
(f) Request for scheduling made by non-preemptive task.



(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 OMAP35x/RVDS-specific parts of RTA-OS can be found in the following manuals:

***OMAP35x/RVDS Release Note*.** This document provides information about the OMAP35x/RVDS port plug-in release, including a list of changes from previous releases and a list of known issues.

Information about the port-independent parts of RTA-OS can be found in the following manuals:

***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 14	Phone:	+49 711 89661-0
70469 Stuttgart	Fax:	+49 711 89661-300
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