# RTA-OSEK

Binding Manual: HC12X/COSMIC

# Contact Details

**ETAS Group**

**www.etasgroup.com**

**Germany**

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

Tel.:+49 (711) 8 96 61-102
Fax:+49 (711) 8 96 61-106

**www.etas.de**

**Japan**

ETAS K.K.
Queen's Tower C-17F,
2-3-5, Minatomirai, Nishi-ku,
Yokohama, Kanagawa
220-6217 Japan

Tel.: +81 (45) 222-0900
Fax: +81 (45) 222-0956

**www.etas.co.jp**

**Korea**

ETAS Korea Co. Ltd.
3F, Samseung Bldg. 61-1
Yangjae-dong, Seocho-gu
Seoul

Tel.: +82 (2) 57 47-016
Fax: +82 (2) 57 47-120

**www.etas.co.kr**

**USA**

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

Tel.: +1 (888) ETAS INC
Fax: +1 (734) 997-94 49

**www.etasinc.com**

**France**

ETAS S.A.S.
1, place des États-Unis
SILIC 307
94588 Rungis Cedex

Tel.: +33 (1) 56 70 00 50
Fax: +33 (1) 56 70 00 51

**www.etas.fr**

**Great Britain**

ETAS UK Ltd.
Studio 3, Waterside Court
Third Avenue, Centrum 100
Burton-upon-Trent
Staffordshire DE14 2WQ

Tel.: +44 (0) 1283 - 54 65 12
Fax: +44 (0) 1283 - 54 87 67

**www.etas-uk.net**

# Copyright Notice

Version: RM00064-01

## Disclaimer

The information in this document is subject to change without notice and does not represent a commitment on any part of LiveDevices. While the information contained herein is assumed to be accurate, LiveDevices assumes no responsibility for any errors or omissions.

In no event shall LiveDevices, its employees, its contractors or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees or expenses of any nature or kind.

## Trademarks

RTA-OSEK and LiveDevices are trademarks of LiveDevices Ltd.

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

All other product names are trademarks or registered trademarks of their respective owners.

# Contents

# 1 About this Guide

This guide provides port specific information for the HC12X/COSMIC implementation of LiveDevices' RTA-OSEK.

A port is defined as a specific target microcontroller/target toolchain pairing. This guide tells you about integration issues with your target toolchain and issues that you need to be aware of when using RTA-OSEK on your target hardware. Port specific parameters of implementation are also provided, giving the RAM and ROM requirements for each object in the RTA-OSEK Component and execution times for each API call to the RTA-OSEK Component.

## 1.1 Who Should Read this Guide?

It is assumed that you are a developer. You should read this guide if you want to know low-level technical information to integrate the RTA-OSEK Component into your application.

## 1.2 Conventions

**Important:** Notes that appear like this contain important information that you need to be aware of. Make sure that you read them carefully and that you follow any instructions that you are given.

**Portability:** Notes that appear like this describe things that you will need to know if you want to write code that will work on any processor running the RTA-OSEK Component.

In this guide you'll see that program code, header file names, C type names, C functions and RTA-OSEK API call names all appear in the `courier` typeface. When the name of an object is made available to the programmer the name also appears in the `courier` typeface, so, for example, a task named Task1 appears as a task handle called `Task1`.

# 2 Toolchain Issues

In this chapter, you'll see the important details that you need to know about RTA-OSEK and your toolchain. A port of the RTA-OSEK Component is specific to both the target hardware *and* the compiler toolchain. You must make sure that you build your application with this toolchain.

If you are interested in using a different version of the same toolchain, you should contact LiveDevices to confirm whether or not this is possible.

## 2.1 Memory Model

The HC12 architecture supports the use of one-byte addresses for the first 256 bytes of the address space, called the "zero page" section or "zpage". Conventionally, however, low memory addresses are used for I/O flags. The RTA-OSEK Component therefore makes no special use of zpage, and the modifier `@dir` is not used.

The HC12 architecture can also support three kinds of memory banking:

- a 3-byte CALL/RTC mechanism in which 16k banks of code are mapped into and out of the space between 0x8000 and 0xBFFF. The PPAGE register stores the index of the currently-mapped bank.
- a banked 4k data space at 0x7000-0x7FFF selectable by writing to the DPAGE register.
- a banked 1k data space in low memory selectable by writing to the EPAGE register.

Only the first of these is explicitly catered for by RTA-OSEK. All library code has been compiled without the `@far` modifier, and must therefore appear in unbanked code. If banked code is used for application code, most of what is required is the user's responsibility. Where the API calls `TerminateTask()` and `WaitEvent()` are used in application code, if a bank switch is required the RTA-OSEK Component makes the necessary modification to PPAGE.

In order to support this, the user is required to make known the location of PPAGE when linking applications, using a linker command of the form

```
+def os_ppage=0x30
```

or whatever the location of PPAGE is on that target. If PPAGE does not exist, the location of any unused byte in RAM should be given.

No support is provided for the use of DPAGE and EPAGE, so these may only be used in ways which do not affect the RTA-OSEK Component in any way.

## 2.2 Compiler

The RTA-OSEK Component was built using the following compiler:

| | |
|---|---|
| Vendor | Cosmic |
| Compiler | cxs12x |
| Version | V4.6a |

The compulsory compiler options for application code are shown in the following table:

| Option | Description |
|---|---|
| `+nowiden` | Do not widen char parameters to integers |

The C file that RTA-OSEK generates from your OIL configuration file is called `osekdefs.c`. This file defines configuration parameters for the RTA-OSEK Component when running your application.

The compulsory compiler options for `osekdefs.c` are shown in the following table:

| Option | Description |
|---|---|
| `+nowiden` | Do not widen char parameters to integers |

## 2.3 Assembler

The RTA-OSEK Component was built using the following assembler:

| | |
|---|---|
| Vendor | Cosmic |
| Assembler | cxs12x |
| Version | V4.6a |

The assembly file that RTA-OSEK generates from your OIL configuration file is called `osgen.s`. This file defines configuration parameters for the RTA-OSEK Component when running your application.

## 2.4 Linker/Locator

The compulsory linker/locator options for an RTA-OSEK application are shown in the following table:

| Option | Description |
|---|---|
| `+def os_ppage=<value>` | <value>=the address of PPAGE if any,<br>or otherwise any unused RAM address |

In addition to the sections used by application code, the following RTA-OSEK sections must be located:

| Sections | Rom/Ram | Description |
|---|---|---|
| `os_pid` | ROM | RTA-OSEK read-only data |
| `os_pird` | ROM | RTA-OSEK initialization data |
| `os_vectbl` | ROM | Relocatable vector table if generated by RTA-OSEK GUI |
| `os_vectbl1` | ROM | Fixed vector table if generated by RTA-OSEK GUI |
| `os_pir` | RAM | RTA-OSEK initialized data |
| `os_pur` | RAM | RTA-OSEK uninitialized data |

All Cosmic run-time libraries are compatible with RTA-OSEK.

## 2.5 Debugger

Information about ORTI for RTA-OSEK can be found in the *RTA-OSEK ORTI Guide*

At the time of writing, we were not aware of any debuggers for the Motorola Star12X with support for ORTI.

If you are using an ORTI version 2.0 aware debugger on this platform you can use the "Unknown ORTI debugger" option in the RTA-OSEK GUI to generate an ORTI output file. The ORTI generated will not have been tested on the debugger and, therefore, is not guaranteed to work.

Please contact LiveDevices if you have any questions about ORTI support in RTA-OSEK.

# 3 Target Hardware Issues

## 3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model. You can find out more about configuring interrupts for RTA-OSEK in the *RTA-OSEK User Guide*.

### 3.1.1 Interrupt Levels

In RTA-OSEK interrupts are allocated an Interrupt Priority Level (IPL). This is a processor independent abstraction of the interrupt priorities that are available on the target hardware. You can find out more about IPLs in the *RTA-OSEK User Guide*. The hardware interrupt controller is explained in the *CPU12X Reference Manual*.

The following table shows how RTA-OSEK IPLs relate to interrupt priorities on the target hardware:

| IPL Value | High Byte Of Condition Code Register | I Bit In Condition Code Register | Description |
|---|---|---|---|
| 0 | 0 | 0 | User level |
| 1 | 1 | 0 | Category 1 and 2 interrupts |
| 2 | 2 | 0 | Category 1 and 2 interrupts |
| 3 | 3 | 0 | Category 1 and 2 interrupts |
| 4 | 4 | 0 | Category 1 and 2 interrupts |
| 5 | 5 | 0 | Category 1 and 2 interrupts |
| 6 | 6 | 0 | Category 1 and 2 interrupts |
| 7 | 7 | 0 | Category 1 and 2 interrupts |
| 8 | any value | 1 | Category 1 interrupts only |

### 3.1.2 Interrupt Vectors

For the allocation of Category 1 and Category 2 interrupt handlers to interrupt vectors on your target hardware, the following restrictions apply:

| Vector | Legality |
|---|---|
| 0xFFFE | RESET cannot be used |
| 0xF4 | use of XIRQ invalidates timing analysis |

The HC12X has some vectors that have a fixed location (vectors 0xFFFC, 0xFFFA) and the rest of the vectors are relocatable. If a vector table is generated it has two sections. os_vectbl1 contains the fixed location vectors and must be located at 0xfffa. os_vectbl contains the relocatable vectors and can be placed at any of the locations outlined in the processor documentation. The user is responsible for initializing the Interrupt Vector Base Register (IVBR).

### 3.1.3 Interrupt Priority Levels

The priority at which a hardware interrupt is taken is set in the INT_CFDATA registers under the control of the INT_CFADDR register.

The RTA-OSEK GUI generates a table (of the interrupt priorities used in the application), called os_InitIrqLevels, which must be used to initialize the INT_CFDATA registers. This table contains the priority levels for interrupts defined in the application.

**Important:** The os_InitIrqLevels table must be copied to the INT_CFDATA registers before the call to StartOS() otherwise interrupts will not work correctly.

The init_target() function in target.c in the example application (located in <RTA-OSEK install directory>\COS12X\Example\) gives an example of how to copy os_InitIrqLevels to the correct location.

### 3.1.4 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Cosmic C compiler can generate appropriate interrupt handling code for a C function decorated with the @interrupt function qualifier. You can find out more in your compiler documentation.

### 3.1.5 Category 2 Handlers

Category 2 ISRs are provided with a C function context by the RTA-OSEK Component, since the RTA-OSEK Component handles the interrupt context itself. The handlers are written using the OSEK OS standard ISR() macro, shown in Code Example 3:1.

```
#include “MyISR.h”
ISR(MyISR) {
  /* Handler routine */
}
```

**Code Example 3:1 - Category 2 ISR Interrupt Handler**

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

### 3.1.6 Vector Table Issues

When you configure your application with the RTA-OSEK GUI you can choose whether or not a vector table is generated within `osgen.s`.

Note that a generated vector table omits the reset vector entry. If you choose to provide your own vector table, it must contain an entry for each interrupt handler, including the Category 2 interrupt handlers in RTA-OSEK.

The following table shows the syntax for labels attached to RTA-OSEK Category 2 interrupt handlers (VV represents the 2 hex digit, upper-case, zero-padded value of the vector location).

| Vector Location | Label |
|---|---|
| 0xVV | `_os_wrapper_VV` |
| e.g. 0x90 | `_os_wrapper_90` |

## 3.2 Register Settings

The RTA-OSEK Component does not require the initialization of registers before calling `StartOS()`.

The RTA-OSEK Component uses the following hardware registers. They should not be altered by user code.

| Registers Used | Notes |
|---|---|
| High byte of CCR | The high byte of the CCR holds the current priority level |
| I bit in the CCR | The I bit enables and disables interrupts. |

## 3.3 Stack Usage

### 3.3.1 Number of Stacks

A single stack is used. The first argument to `StackFaultHook` is always 0.

`StackOffsetType` is a scalar, representing the number of bytes on the stack, with C type: `unsigned short`

### 3.3.2 Stack Usage within API Calls

The maximum stack usage within RTA-OSEK API calls, excluding calls to hooks and callbacks, is as follows:

### Standard

API max usage (bytes): 15

### Timing

API max usage (bytes): 15

### Extended

API max usage (bytes): 23

To determine the correct stack usage for tasks that use other library code, you may need to contact the vendor to find out more about library call stack usage.

# 4 Parameters of Implementation

This chapter provides detailed information on the functionality, performance and memory demands of the RTA-OSEK Component.

NB: This is a placeholder for the tables of sizes and times collected by the Binding Manual Performance Measurement application. At the time of generation of this manual, this application is not yet available for the HC12X/COSMIC port of RTA-OSEK.

# Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found on the ETAS Group website www.etasgroup.com.