RTA-OSEK

Binding Manual: Tasking/167

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: RM00007-003

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 Tasking/167 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 RTA-OSEK runtime libraries have been built for the small memory model. The OS code in the "`OS_WRAPPER_CLASS`" class must be located within a single code segment.

All externally visible (API) OS functions can be called from user programs in any code segment.

2.2 Compiler

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

Vendor	Tasking
Compiler	C
Version	v7.5 r1

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

Option	**Description**
`-Ms`	Small memory model
`-x`	Extended C167 instructions

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

Option	**Description**
`-Ob`	Uninitialized memory is not zeroed

The Tasking supplied C startup, `start.asm`, can be used unmodified. It must be processed with the Tasking macro-preprocessor "`m166`" to select the small memory model and the extended architecture using the "`DEFINE(MODEL,SMALL) DEFINE(_EXT,1)`" options. It must then be assembled with "`a166`" using the "`EXTSFR EXTSSK EXTMEM EXTPEC`" options. This is demonstrated in the example application, which is discussed in more detail in the *RTA-OSEK Getting Started Guide*.

2.3 Assembler

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

Vendor	Tasking
Assembler	C166/ST10 assembler
Version	v7.5 r1

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

2.4 Linker/Locator

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	Vector table if generated by RTA-OSEK GUI
`os_pir`	RAM	RTA-OSEK initialized data
`os_pur`	RAM	RTA-OSEK uninitialized data

The following compiler run-time library functions are required by the RTA-OSEK Component:

C Library Functions	Description
`setjmp`	ISO C library function
`longjmp`	ISO C library function

This port of the RTA-OSEK Component is compatible with Tasking's "extended" set of run-time libraries for the small memory model except for those optimized for MAC instructions. The following table lists the compatible libraries:

Library	Description (See Tasking compiler User's Guide for details)
`ext\c166s.lib`	C library
`ext\c166ss.lib`	C library for single precision floating point
`ext\fp166s.lib`	Floating point library
`ext\fp166st.lib`	Floating point library with trapping

`ext\rt166s.lib`	Run-time library
`ext\rt166ss.lib`	Run-time library for single precision floating point

2.5 Debugger

ORTI is the OSEK Run-Time Interface that is supported by RTA-OSEK. Support is provided for the debuggers in the following table. Further information about ORTI for RTA-OSEK can be found in the *RTA-OSEK ORTI Guide*.

ORTI compatible debuggers	Crossview Pro C166/ST10 v7.5r1, OSEK/ORTI v2.0, RADM 011

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 *[X]C16x User's Manual*.

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

IPL Value	ILVL Value	Description
0	0000	User Level
1-13	0001-1101	Category 1 and 2 interrupts
14-15	1110-1111	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
0x2-0x7F	Category 1
0x3,0x5,0x7,0x8,0x9,0xB-0x47	Category 2

The valid base addresses for the vector table are:

Base Address	Notes
0x0	

3.1.3 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Tasking 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.

3.1.4 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.

3.1.5 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.src`.

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 (VVVV represents the 4 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
`0xVVVV`	`_os_wrapper_VVVV`
e.g. `0x004C`	`_os_wrapper_004C`

The vector table generated by the RTA-OSEK GUI locates itself at address 0x8, leaving the reset vector at address 0x0 unaffected, and does not scale.

Even when the RTA-OSEK GUI does not generate a vector table, suitable directives are included in `osgen.src` to allow the Tasking linker/locator to build a vector table that includes correct entries for the declared Category 2 interrupts. We recommend using this method of building a vector table.

3.2 Exiting Interrupts

Important: When exiting an interrupt handler using the RETI instruction, the 16x family of processors will execute two cycles of the interrupted code before handling a pending interrupt (Infineon *C167 User’s Manual*).

Because of this feature, interrupt handlers must exit with care to avoid unexpectedly running low priority code when interrupts are pending. For a fuller explanation, see Section 5

RTA-OSEK exits interrupts correctly for all Category 2 interrupts. Category 1 interrupt handlers must be written so that these additional two cycles are consumed within the interrupt handler. In order to implement this behavior the following assembly fragment can be used, which causes the terminal `reti` instruction to execute twice, and thus, consume the two cycles.

```
 …
 mov R1, SP               ; Get stack pointer
 add R1, #4               ; Offset to stacked PSW
 mov R1, [R1]             ; Read stacked PSW
 bmov R1.5, MULIP         ; Preserve current MULIP
                          ; at first reti
 push R1                  ; Push PSW(MULIP) onto stack
 mov R1, #SEG _reti       ; Force reti to execute
twice
 push R1
 mov R1, #_reti
 push R1
 ; restore registers, including R1
 …
_reti: LABEL FAR
 reti                     ; This instruction
                          ; executes twice
```

3.3 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.

PSW Field	**Notes**
ILVL	Used to control IPL
IEN	Must not be cleared
MULIP	Must not be directly manipulated by user code

3.4 Stack Usage

3.4.1 Number of Stacks

Two stacks are used. The System stack is indexed 0 and the User stack is indexed 1.

The first argument to `StackFaultHook` is 0 or 1. This indicates the stack on which the excess was observed.

`StackOffsetType` is a structure of two scalars, representing the number of bytes on each stack. This is shown in Code Example 3:2.

```
typedef unsigned short  osStackBytes0;
typedef unsigned short  osStackBytes1;
typedef struct {
        osStackBytes0   sys;
        osStackBytes1   usr;
}                       StackOffsetType;
```

Code Example 3:2 - StackOffsetType()

3.4.2 Stack Usage within API Calls

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

Standard

Stack	API Max Usage (Bytes)
System stack	8
User stack (C stack)	6

Timing

Stack	API Max Usage (Bytes)
System stack	8
User stack (C stack)	6

Extended

Stack	API Max Usage (Bytes)
System stack	8
User stack (C stack)	18

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.

The RTA-OSEK Component is highly scalable. As a result, different figures will be obtained when your application uses different sets of features. These feature-sets give six classes of RTA-OSEK, depending on whether your application uses events, shared task priorities and/or multiple (queued) task activations. You should identify which class your application belongs to and then use the figures from the appropriate column in the table.

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

Processor	**C167**
Clock speed (MHz)	20
Code memory	External RAM
Read-only data memory	Internal RAM
Read-write data memory	Internal RAM

4.1 Functionality

The OSEK Operating System Specification specifies four conformance classes. These attributes apply to *systems* built with OSEK OS objects. The following table specifies the number of OSEK OS and COM objects supported per conformance class.

Configuration	**Application Uses**					
Events	**No**			**Yes**		
Shared Task Priorities	**No**		**Yes**	**No**		**Yes**
Multiple Task Activations	**No**	**Yes**		**No**	**Yes**	
Maximum number of tasks	16	16	16	16	16	16
Maximum number of not suspended tasks	16	16	16	16	16	16
Maximum number of priorities	16	16	16	16	16	16
Number of tasks per priority (for BCC2 and ECC2)	n/a	16	16	n/a	16	16
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	16	16	16
Limits for the number of alarm objects (per system / per task)	not limited by RTA-OSEK					
Limits for the number of standard resources (per system)	255	255	255	255	255	255
Limits for the number of internal resources (per system)	not limited by RTA-OSEK					
Limits for the number of nested resources (per system / per task)	255	255	255	255	255	255
Limits for the number of application modes	255					

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
(per system)						

4.2 Hardware Resources

4.2.1 ROM and RAM Overheads

The following tables give the ROM and RAM overheads for the RTA-OSEK Component (in bytes). The OSEK COM overheads are quoted separately. If you do not use messages, your application will not include this overhead for the parts of OSEK COM required to implement messaging.

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	12	12	12	12	12	12
	ROM	99	99	99	99	99	99
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	22	22	22	22	22	22
	ROM	135	135	135	135	135	135
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	30	30	30	30	30	30
	ROM	155	155	155	155	155	155
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

4.2.2 ROM and RAM for OSEK OS Objects

In addition to the base OS overhead, detailed in Section 4.2.1, each OSEK OS object requires ROM and/or RAM. RTA-OSEK provides additional sub-task types for each task type in OSEK (basic and extended), determined by the offline configuration tools. They are as follows:

OSEK Class	Termination	Arithmetic
BCC1	Lightweight	Integer or Floating-Point
BCC1	Heavyweight	Integer or Floating-Point
BCC2	Light or Heavy	Integer or Floating-Point
ECC1	Heavyweight	Integer
ECC1	Heavyweight	Floating-Point
ECC2	Heavyweight	Integer
ECC2	Heavyweight	Floating-Point

The following tables give the ROM and/or RAM requirements (in bytes) for each OS object in the RTA-OSEK Component. (Note that the OSEK COM class was set to CCCA for systems without events, CCCB for systems with events. A default message of size 10 bytes was used for both CCCA and CCCB. The CCCB message size includes queued messages.)

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
BCC1 Heavyweight task	RAM	2	2	2	2	2	2
	ROM	26	26	26	26	26	26
BCC2 task	RAM	n/a	4	6	n/a	4	6
	ROM	n/a	28	32	n/a	28	32
ECC1, Integer task	RAM	n/a	n/a	n/a	22	22	22
	ROM	n/a	n/a	n/a	42	42	42
ECC1, floating-point task	RAM	n/a	n/a	n/a	24	24	24
	ROM	n/a	n/a	n/a	42	42	42
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	24
	ROM	n/a	n/a	n/a	n/a	n/a	46
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	26
	ROM	n/a	n/a	n/a	n/a	n/a	46
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	34	34	34	34	34	34
Category 2 ISR, floating-point	RAM	2	2	2	2	2	2
	ROM	58	58	58	58	58	58
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Alarm	RAM	6	6	6	6	6	6
	ROM	28	28	28	28	28	28
Counter	RAM	2	2	2	2	2	2
	ROM	26	26	26	26	26	26
Message	RAM	11	11	11	51	51	51
	ROM	12	12	12	30	30	30
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	6	0	6	6
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	8	8	8	8	8	8
	ROM	26	26	26	26	26	26
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	6	6	6	6	6	6
	ROM	30	30	30	30	30	30
BCC1 Heavyweight task	RAM	8	8	8	8	8	8
	ROM	32	32	32	32	32	32
BCC2 task	RAM	n/a	10	12	n/a	10	12
	ROM	n/a	34	38	n/a	34	38
ECC1, Integer task	RAM	n/a	n/a	n/a	28	28	28
	ROM	n/a	n/a	n/a	48	48	48
ECC1, floating-point task	RAM	n/a	n/a	n/a	30	30	30
	ROM	n/a	n/a	n/a	48	48	48
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	30
	ROM	n/a	n/a	n/a	n/a	n/a	52
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	32
	ROM	n/a	n/a	n/a	n/a	n/a	52

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	6	6	6	6	6	6
	ROM	56	56	56	56	56	56
Category 2 ISR, floating-point	RAM	8	8	8	8	8	8
	ROM	66	66	66	66	66	66
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Alarm	RAM	6	6	6	6	6	6
	ROM	28	28	28	28	28	28
Counter	RAM	2	2	2	2	2	2
	ROM	26	26	26	26	26	26
Message	RAM	11	11	11	51	51	51
	ROM	12	12	12	30	30	30
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	6	0	6	6
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	8	8	8	8	8	8
	ROM	26	26	26	26	26	26
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	8	8	8	8	8	8
	ROM	34	34	34	34	34	34
BCC1 Heavyweight task	RAM	10	10	10	10	10	10
	ROM	34	34	34	34	34	34
BCC2 task	RAM	n/a	12	14	n/a	12	14
	ROM	n/a	36	40	n/a	36	40
ECC1, Integer task	RAM	n/a	n/a	n/a	30	30	30
	ROM	n/a	n/a	n/a	50	50	50
ECC1, floating-point task	RAM	n/a	n/a	n/a	32	32	32
	ROM	n/a	n/a	n/a	50	50	50
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	32
	ROM	n/a	n/a	n/a	n/a	n/a	54
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	34
	ROM	n/a	n/a	n/a	n/a	n/a	54
Category 2 ISR	RAM	8	8	8	8	8	8
	ROM	60	60	60	60	60	60
Category 2 ISR, floating-point	RAM	10	10	10	10	10	10
	ROM	70	70	70	70	70	70
Resource	RAM	4	4	4	4	4	4
	ROM	14	14	14	14	14	14
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	4	4	4	4	4	4
	ROM	14	14	14	14	14	14
Alarm	RAM	6	6	6	6	6	6
	ROM	30	30	30	30	30	30
Counter	RAM	2	2	2	2	2	2
	ROM	28	28	28	28	28	28
Message	RAM	11	11	11	51	51	51
	ROM	14	14	14	32	32	32
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	4	4	4	4	4	4
	ROM	14	14	14	14	14	14

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	6	0	6	6
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (writable)	RAM	10	10	10	10	10	10
	ROM	10	10	10	10	10	10
Schedule	RAM	10	10	10	10	10	10
	ROM	30	30	30	30	30	30
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

4.2.3 Size of Linkable Modules

The RTA-OSEK Component is demand linked. This means that each API call is placed into a separately linkable module. The following sections list the module sizes (in bytes) for each API call in the 3 RTA-OSEK build types (standard, timing, and extended).

In some cases there are multiple variants of particular API calls. This is because the offline configuration of RTA-OSEK can determine when optimized versions of the API calls can be used. The smallest and fastest call will be selected. In these cases, modules sizes are given for each variant under the particular configuration of the RTA-OSEK Component for which the call is valid.

The call variants are as follows:

Variant	Description
1i	Idle task is only ECC task.
CCCA	OSEK COM class.
CCCB	OSEK COM class.
CLEx	Resource tests in Extended OS Status.
fp	ECC task uses floating-point.
H	Used for heavyweight termination only.

Variant	Description
Hook	Pre- and Post- Task hooks are used.
KL	API is called from OS level.
KL1i	API is called from OS level, idle task is only ECC task.
KL2	Activated taskset has one BCC2 task.
LExt	Used for lightweight termination in Extended Status.
ServiceID	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
Parameters	`ErrorHook` uses `GetServiceID` and `GetServiceParameters`.
NoHook	Pre- and/or Post- Task hooks are not used.
NS	No context switch is possible.
NS1i	No context switch is possible, idle task is only ECC task.
NS2	Activated taskset has one BCC2 task.
NSH	Chain from heavyweight task, not to higher priority.
NSL	Chain from lightweight task, not to higher priority.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.
Task	Resource is used only by tasks.

Standard

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	94	160	222	98	164	242
	NS		72	140	202	76	144	222
	KL	2	42	108	170	46	112	190

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	16	16	16	16	16	16
ChainTask	SWL	1, 8	72	140	200	76	144	220
	SWH	1, 9	96	170	230	100	174	250
	NSL	8	72	140	200	76	144	220
	NSH	9	92	166	226	96	170	246
Schedule			70	70	104	70	70	104
GetTaskID			26	26	26	26	26	26
GetTaskState			52	52	52	68	68	68
EnableAllInterrupts			18	18	18	18	18	18
DisableAllInterrupts			26	26	26	26	26	26
ResumeAllInterrupts			30	30	30	30	30	30
SuspendAllInterrupts			51	51	51	51	51	51
ResumeOSInterrupts			32	32	32	32	32	32
SuspendOSInterrupts			55	55	55	55	55	55
GetResource	Task	7	20	20	26	20	20	26
	Combined	6	70	70	70	70	70	70
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	52	52	52	52	52	52
	Combined	6	130	130	130	130	130	130
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	98	98	206
	NS		n/a	n/a	n/a	74	74	178
	NS1i	10	n/a	n/a	n/a	38	n/a	n/a
	KL	2	n/a	n/a	n/a	56	56	160
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	36	36	36
GetEvent			n/a	n/a	n/a	14	14	14
WaitEvent	<default>		n/a	n/a	n/a	268	268	518
	fp	11	n/a	n/a	n/a	308	308	600
	1i	10	n/a	n/a	n/a	18	n/a	n/a
GetAlarmBase			40	40	40	40	40	40
GetAlarm			70	70	70	70	70	70
SetRelAlarm			78	78	78	78	78	78
SetAbsAlarm			102	102	102	102	102	102
CancelAlarm			68	68	68	68	68	68
InitCounter			46	46	46	46	46	46

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			58	58	58	58	58	58
osek_tick_alarm	<default>		64	64	64	64	64	64
	KL	2	42	42	42	42	42	42
osek_incr_counter			46	46	46	46	46	46
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			92	92	92	92	92	92
ShutdownOS	NoHook	12	24	24	24	24	24	24
	Hook	13	32	32	32	32	32	32
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			20	20	20	20	20	20
StopCOM			16	16	16	16	16	16
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	56	56	56	196	196	196
	CCCB	15	196	196	196	196	196	196
GetMessageResource			32	32	32	32	32	32
ReleaseMessageResource			28	28	28	28	28	28
GetMessageStatus			58	58	58	58	58	58
SendMessage	SW CCCA	1, 14	98	98	98	272	272	272
	SW CCCB	1, 15	252	252	252	272	272	272
	NS CCCA	14	98	98	98	272	272	272
	NS CCCB	15	252	252	252	272	272	272
	KL CCCA	2, 14	78	78	78	254	254	254
	KL CCCB	2, 15	232	232	232	254	254	254
main_dispatch	NoHook	12	106	106	144	106	106	144
	Hook	13	138	138	176	138	138	176
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	78	78	78	78	78	78
	B1HF	21	86	86	86	86	86	86
	B2LI	22	n/a	56	96	n/a	56	96
	B2LF	23	n/a	62	102	n/a	62	102
	B2HI	24	n/a	130	238	n/a	130	238
	B2HF	25	n/a	138	246	n/a	138	246
	E1HI	26	n/a	n/a	n/a	290	290	398
	E1HF	27	n/a	n/a	n/a	298	298	406
	E2HI	28	n/a	n/a	n/a	n/a	n/a	398

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	E2HF	29	n/a	n/a	n/a	n/a	n/a	406
ErrorHook support		16	26	26	26	26	26	26
	ServiceID	17	32	32	32	32	32	32
	Parameters	18	44	44	44	44	44	44
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	n/a	n/a	n/a	n/a	n/a	n/a
Timing_termination		4	n/a	n/a	n/a	n/a	n/a	n/a
ActivateTaskset	SW	1	52	90	156	60	106	176
	NS		28	66	132	36	82	152
	KL	2	10	48	114	18	64	134
ChainTaskset	SWL	1, 8	32	70	136	32	78	148
	SWH	1, 9	54	102	168	54	110	180
	NSL	8	32	70	136	32	78	148
	NSH	9	50	98	164	50	106	176
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			30	30	30	30	30	30
AssignTaskset			6	6	6	6	6	6
RemoveTaskset			32	32	32	32	32	32
TestSubTaskset			42	42	42	42	42	42
TestEquivalentTaskset			38	38	38	38	38	38
TickSchedule	SW	1	126	136	136	136	136	136
	NS		102	116	116	116	116	116
	KL	2	84	94	94	94	94	94
AdvanceSchedule	SW	1	112	118	118	118	118	118
	NS		92	98	98	98	98	98
	KL	2	70	76	76	76	76	76
StartSchedule			70	70	70	70	70	70
StopSchedule			52	52	52	52	52	52
GetScheduleStatus			86	86	86	86	86	86
GetScheduleValue			60	60	60	60	60	60
GetScheduleNext			10	10	10	10	10	10
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			10	10	10	10	10	10
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			8	8	8	8	8	8

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestArrivalpointWritable			36	36	36	36	36	36
GetExecutionTime			4	4	4	4	4	4
GetLargestExecutionTime			8	8	8	8	8	8
ResetLargestExecutionTime			4	4	4	4	4	4
GetStackOffset			28	28	28	28	28	28
Floating point support			28	28	28	28	28	28
Interrupt support			184	184	184	184	184	184
Utility functions			62	62	62	62	62	62

Timing

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	94	160	222	98	164	242
	NS		72	140	202	76	144	222
	KL	2	42	108	170	46	112	190
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	16	16	16	16	16	16
ChainTask	SWL	1, 8	72	140	200	76	144	220
	SWH	1, 9	96	170	230	100	174	250
	NSL	8	72	140	200	76	144	220
	NSH	9	92	166	226	96	170	246
Schedule			78	78	112	78	78	112
GetTaskID			26	26	26	26	26	26
GetTaskState			52	52	52	68	68	68
EnableAllInterrupts			18	18	18	18	18	18
DisableAllInterrupts			26	26	26	26	26	26
ResumeAllInterrupts			30	30	30	30	30	30
SuspendAllInterrupts			51	51	51	51	51	51
ResumeOSInterrupts			32	32	32	32	32	32
SuspendOSInterrupts			55	55	55	55	55	55
GetResource	Task	7	20	20	26	20	20	26
	Combined	6	70	70	70	70	70	70

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	60	60	60	60	60	60
	Combined	6	146	146	146	146	146	146
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	98	98	206
	NS		n/a	n/a	n/a	74	74	178
	NS1i	10	n/a	n/a	n/a	38	n/a	n/a
	KL	2	n/a	n/a	n/a	56	56	160
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	36	36	36
GetEvent			n/a	n/a	n/a	14	14	14
WaitEvent	<default>		n/a	n/a	n/a	298	298	522
	fp	11	n/a	n/a	n/a	332	332	592
	1i	10	n/a	n/a	n/a	74	n/a	n/a
GetAlarmBase			40	40	40	40	40	40
GetAlarm			70	70	70	70	70	70
SetRelAlarm			78	78	78	78	78	78
SetAbsAlarm			102	102	102	102	102	102
CancelAlarm			68	68	68	68	68	68
InitCounter			46	46	46	46	46	46
GetCounterValue			58	58	58	58	58	58
osek_tick_alarm	<default>		64	64	64	64	64	64
	KL	2	42	42	42	42	42	42
osek_incr_counter			46	46	46	46	46	46
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			128	128	128	128	128	128
ShutdownOS	NoHook	12	24	24	24	24	24	24
	Hook	13	32	32	32	32	32	32
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			20	20	20	20	20	20
StopCOM			16	16	16	16	16	16
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	56	56	56	196	196	196
	CCCB	15	196	196	196	196	196	196
GetMessageResource			32	32	32	32	32	32

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
ReleaseMessageResource			28	28	28	28	28	28
GetMessageStatus			58	58	58	58	58	58
SendMessage	SW CCCA	1, 14	98	98	98	272	272	272
	SW CCCB	1, 15	252	252	252	272	272	272
	NS CCCA	14	98	98	98	272	272	272
	NS CCCB	15	252	252	252	272	272	272
	KL CCCA	2, 14	78	78	78	254	254	254
	KL CCCB	2, 15	232	232	232	254	254	254
main_dispatch	NoHook	12	160	160	198	160	160	198
	Hook	13	196	196	234	196	196	234
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	82	82	82	82	82	82
	B1HF	21	90	90	90	90	90	90
	B2LI	22	n/a	36	76	n/a	36	76
	B2LF	23	n/a	42	82	n/a	42	82
	B2HI	24	n/a	112	220	n/a	112	220
	B2HF	25	n/a	120	228	n/a	120	228
	E1HI	26	n/a	n/a	n/a	304	304	412
	E1HF	27	n/a	n/a	n/a	312	312	420
	E2HI	28	n/a	n/a	n/a	n/a	n/a	412
	E2HF	29	n/a	n/a	n/a	n/a	n/a	420
ErrorHook support		16	26	26	26	26	26	26
	ServiceID	17	32	32	32	32	32	32
	Parameters	18	44	44	44	44	44	44
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	58	58	58	58	58	58
Timing_termination		4	66	66	66	66	66	66
ActivateTaskset	SW	1	52	90	156	60	106	176
	NS		28	66	132	36	82	152
	KL	2	10	48	114	18	64	134
ChainTaskset	SWL	1, 8	32	70	136	32	78	148
	SWH	1, 9	54	102	168	54	110	180
	NSL	8	32	70	136	32	78	148
	NSH	9	50	98	164	50	106	176
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			30	30	30	30	30	30
AssignTaskset			6	6	6	6	6	6

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
RemoveTaskset			32	32	32	32	32	32
TestSubTaskset			42	42	42	42	42	42
TestEquivalentTaskset			38	38	38	38	38	38
TickSchedule	SW	1	126	136	136	136	136	136
	NS		102	116	116	116	116	116
	KL	2	84	94	94	94	94	94
AdvanceSchedule	SW	1	112	118	118	118	118	118
	NS		92	98	98	98	98	98
	KL	2	70	76	76	76	76	76
StartSchedule			70	70	70	70	70	70
StopSchedule			52	52	52	52	52	52
GetScheduleStatus			86	86	86	86	86	86
GetScheduleValue			60	60	60	60	60	60
GetScheduleNext			10	10	10	10	10	10
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			10	10	10	10	10	10
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			8	8	8	8	8	8
TestArrivalpointWritable			36	36	36	36	36	36
GetExecutionTime			70	70	70	70	70	70
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			14	14	14	14	14	14
GetStackOffset			28	28	28	28	28	28
Floating point support			28	28	28	28	28	28
Interrupt support			164	164	164	164	164	164
Utility functions			62	62	62	62	62	62
Interrupt support			118	118	118	118	118	118

Extended

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	214	280	340	218	284	360
	NS		272	338	398	276	342	418
	KL	2	128	200	262	132	204	282
TerminateTask	LExt	3	96	96	96	96	96	96
	H	5	126	126	126	126	126	126
ChainTask	SWL	1, 8	234	306	366	238	310	388
	SWH	1, 9	270	342	402	274	346	422
	NSL	8	314	380	440	318	384	460
	NSH	9	346	412	472	350	416	492
Schedule			196	196	230	196	196	230
GetTaskID			38	38	38	38	38	38
GetTaskState			196	196	196	198	198	198
EnableAllInterrupts			30	30	30	30	30	30
DisableAllInterrupts			42	42	42	42	42	42
ResumeAllInterrupts			76	76	76	76	76	76
SuspendAllInterrupts			67	67	67	67	67	67
ResumeOSInterrupts			78	78	78	78	78	78
SuspendOSInterrupts			71	71	71	71	71	71
GetResource	Task	7	348	348	310	348	348	310
	Combined	6	334	334	334	334	334	334
	CLEx	3	290	290	290	290	290	290
ReleaseResource	Task	7	306	306	306	306	306	306
	Combined	6	378	378	378	378	378	378
	CLEx	3	292	292	292	292	292	292
SetEvent	SW	1	n/a	n/a	n/a	278	278	380
	NS		n/a	n/a	n/a	338	338	438
	NS1i	10	n/a	n/a	n/a	212	n/a	n/a
	KL	2	n/a	n/a	n/a	186	186	296
	KL1i	2, 10	n/a	n/a	n/a	140	n/a	n/a
ClearEvent			n/a	n/a	n/a	116	116	116
GetEvent			n/a	n/a	n/a	134	134	134
WaitEvent	<default>		n/a	n/a	n/a	422	422	630
	fp	11	n/a	n/a	n/a	460	460	708

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	1i	10	n/a	n/a	n/a	196	n/a	n/a
GetAlarmBase			164	164	164	164	164	164
GetAlarm			150	150	150	150	150	150
SetRelAlarm			202	202	202	202	202	202
SetAbsAlarm			232	232	232	232	232	232
CancelAlarm			140	140	140	140	140	140
InitCounter			164	164	164	164	164	164
GetCounterValue			188	188	188	188	188	188
osek_tick_alarm	<default>		106	106	106	106	106	106
	KL	2	42	42	42	42	42	42
osek_incr_counter			46	46	46	46	46	46
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			144	144	144	144	144	144
ShutdownOS	NoHook	12	32	32	32	32	32	32
	Hook	13	40	40	40	40	40	40
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			36	36	36	36	36	36
StopCOM			38	38	38	38	38	38
ReadFlag			28	28	28	28	28	28
ResetFlag			30	30	30	30	30	30
ReceiveMessage	CCCA	14	160	160	160	298	298	298
	CCCB	15	298	298	298	298	298	298
GetMessageResource			84	84	84	84	84	84
ReleaseMessageResource			84	84	84	84	84	84
GetMessageStatus			102	102	102	102	102	102
SendMessage	SW CCCA	1, 14	206	206	206	374	374	374
	SW CCCB	1, 15	354	354	354	374	374	374
	NS CCCA	14	206	206	206	374	374	374
	NS CCCB	15	354	354	354	374	374	374
	KL CCCA	2, 14	154	154	154	328	328	328
	KL CCCB	2, 15	308	308	308	328	328	328
main_dispatch	NoHook	12	160	160	198	160	160	198
	Hook	13	196	196	234	196	196	234
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	84	84	84	84	84	84
	B1HF	21	92	92	92	92	92	92

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	B2LI	22	n/a	36	76	n/a	36	76
	B2LF	23	n/a	42	82	n/a	42	82
	B2HI	24	n/a	114	222	n/a	114	222
	B2HF	25	n/a	122	230	n/a	122	230
	E1HI	26	n/a	n/a	n/a	308	308	416
	E1HF	27	n/a	n/a	n/a	316	316	424
	E2HI	28	n/a	n/a	n/a	n/a	n/a	416
	E2HF	29	n/a	n/a	n/a	n/a	n/a	424
ErrorHook support		16	92	92	92	92	92	92
	ServiceID	17	98	98	98	98	98	98
	Parameters	18	110	110	110	110	110	110
validity_checks		3	24	24	24	24	24	24
Timing_dispatch		4	58	58	58	58	58	58
Timing_termination		4	66	66	66	66	66	66
ActivateTaskset	SW	1	326	368	432	340	390	464
	NS		386	428	492	400	450	524
	KL	2	246	288	352	260	310	384
ChainTaskset	SWL	1, 8	366	408	474	370	420	496
	SWH	1, 9	406	456	522	410	468	544
	NSL	8	444	486	552	448	498	574
	NSH	9	480	530	596	484	542	618
GetTasksetRef			106	106	106	106	106	106
MergeTaskset			310	310	310	310	310	310
AssignTaskset			230	230	230	230	230	230
RemoveTaskset			312	312	312	312	312	312
TestSubTaskset			322	322	322	322	322	322
TestEquivalentTaskset			318	318	318	318	318	318
TickSchedule	SW	1	336	272	272	272	272	272
	NS		404	370	370	370	370	370
	KL	2	272	210	210	210	210	210
AdvanceSchedule	SW	1	336	276	276	276	276	276
	NS		404	368	368	368	368	368
	KL	2	282	222	222	222	222	222
StartSchedule			252	252	252	252	252	252
StopSchedule			196	196	196	196	196	196
GetScheduleStatus			234	234	234	234	234	234
GetScheduleValue			188	188	188	188	188	188

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleNext			96	96	96	96	96	96
SetScheduleNext			216	216	216	216	216	216
GetArrivalpointDelay			154	154	154	154	154	154
SetArrivalpointDelay			172	172	172	172	172	172
GetArrivalpointTasksetRef			150	150	150	150	150	150
GetArrivalpointNext			154	154	154	154	154	154
SetArrivalpointNext			256	256	256	256	256	256
TestArrivalpointWritable			176	176	176	176	176	176
GetExecutionTime			112	112	112	112	112	112
GetLargestExecutionTime			92	92	92	92	92	92
ResetLargestExecutionTime			90	90	90	90	90	90
GetStackOffset			28	28	28	28	28	28
Floating point support			28	28	28	28	28	28
Interrupt support			164	164	164	164	164	164
Utility functions			62	62	62	62	62	62
Interrupt support			118	118	118	118	118	118

Notes

Number	Note
1	Linked only if upward activations are allowed
2	Linked only if API is called within ISR
3	Present only in Extended OS status
4	Present only in Timing or Extended OS status
5	Linked only if there are heavyweight tasks in the system
6	Linked only if Resource is used by both tasks and ISRs
7	Linked only if Resource is used only by tasks
8	Linked only if Chaining task is Lightweight
9	Linked only if Chaining task is Heavyweight
10	Linked only if Idle task is the only extended task in the system
11	Linked only if calling Extended task uses floating-point
12	Linked only if neither Pre- nor Post-TaskHook is used
13	Linked only if Pre- or Post-TaskHook is used
14	Linked only if there are no flags, message queues, or message resources in the system, and COM status is not requested.
15	Linked only if there are any flags, message queues, or message resources in the system, or COM status is requested.

Number	Note
16	Linked only if `USEGETSERVICEID = FALSE` and `USEPARAMETERACCESS = FALSE`
17	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = FALSE`
18	Linked only if `USEGETSERVICEID = TRUE` and USEPARAMETERACCESS = TRUE
19	Linked only for basic, single-activation, lightweight, floating-point tasks
20	Linked only for basic, single-activation, heavyweight, integer tasks
21	Linked only for basic, single-activation, heavyweight, floating-point tasks
22	Linked only for basic, multiple-activation, lightweight, integer tasks
23	Linked only for basic, multiple-activation, lightweight, floating-point tasks
24	Linked only for basic, multiple-activation, heavyweight, integer tasks
25	Linked only for basic, multiple-activation, heavyweight, floating-point tasks
26	Linked only for extended, unique priority, integer tasks
27	Linked only for extended, unique priority, floating-point tasks
28	Linked only for extended, shared priority, integer tasks
29	Linked only for extended, shared priority, floating-point tasks
30	Implemented as a macro, so no code is linked
31	Not required on some targets

4.2.4 Reserved Hardware Resources

Timer units, interrupts, traps and other hardware resources are not reserved by RTA-OSEK.

4.3 Performance

The collection of performance data for the Tasking/167 port of the RTA-OSEK Component was achieved using a timer running four times slower than the CPU clock speed. The figures in this section, therefore, have an uncertainty level of up to four CPU cycles. The actual times are between 0 and four cycles shorter than those reported in the remainder of this section.

4.3.1 Execution Times for RTA-OSEK API Calls

The following tables give the execution time (in CPU cycles) for each API call. (Note that: (1) the OSEK COM class was set to CCCA for systems without events and to CCCB for systems with events; (2) `ShutdownOS()` enters an infinite loop; the execution time for `ShutdownOS()` reported below is the time up to the point at which `ShutdownOS()` calls `ShutdownHook()`).

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	164	264	392	176	240	400
	NS	120	232	360	132	208	368
	KL	84	180	308	92	156	316
TerminateTask	LExt	0	0	0	0	0	0
	H	292	288	292	292	292	292
ChainTask	SWL	492	592	800	672	740	984
	SWH	656	760	972	836	912	1156
	NSL	492	588	800	676	740	984
	NSH	644	748	960	824	904	1148
Schedule	SW	144	136	180	144	144	180
GetTaskID		76	72	76	72	76	76
GetTaskState		124	120	124	152	152	152
EnableAllInterrupts		48	44	48	48	48	48
DisableAllInterrupts		64	60	64	64	68	68
ResumeAllInterrupts		76	72	76	76	72	72
SuspendAllInterrupts		92	88	92	92	92	92
ResumeOSInterrupts		76	72	76	76	72	72
SuspendOSInterrupts		92	88	92	92	92	92
GetResource	Task	80	76	88	80	80	88
	Combined	112	108	112	112	112	112
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	112	108	112	112	112	112
	Combined	152	152	156	152	152	156
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	168	168	172
	NS	n/a	n/a	n/a	168	168	168
	KL	n/a	n/a	n/a	140	140	144
ClearEvent		n/a	n/a	n/a	96	96	92
GetEvent		n/a	n/a	n/a	60	60	60
WaitEvent	<default>	n/a	n/a	n/a	960	960	1076
	fp	n/a	n/a	n/a	992	992	1108
GetAlarmBase		124	120	124	124	124	124
GetAlarm		144	144	144	148	144	144

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		172	168	168	168	168	172
SetAbsAlarm		188	180	184	188	184	184
CancelAlarm		124	124	128	128	128	128
InitCounter		104	104	104	108	104	108
GetCounterValue		108	104	112	112	112	108
osek_tick_alarm	<default>	164	160	164	164	164	164
	KL	108	108	112	112	112	112
osek_incr_counter		44	40	44	44	44	44
GetActiveApplicationMode		20	20	24	24	24	24
StartOS		1148	1152	1148	1148	1148	1148
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	68	64	68	68	68	68
InitCOM		32	28	28	32	32	32
CloseCOM		28	24	28	32	32	32
StartCOM		76	72	80	376	372	372
StopCOM		52	52	56	52	52	56
ReadFlag		n/a	n/a	n/a	40	44	40
ResetFlag		n/a	n/a	n/a	32	32	36
ReceiveMessage		88	84	88	500	500	496
GetMessageResource		n/a	n/a	n/a	184	184	188
ReleaseMessageResource		n/a	n/a	n/a	208	208	208
GetMessageStatus		n/a	n/a	n/a	136	136	136
SendMessage	SW	300	396	524	732	800	956
	NS	248	356	488	680	760	920
	KL	172	272	396	620	688	844
ActivateTaskset	SW	116	584	744	132	596	768
	NS	48	544	696	92	556	716
	KL	48	512	664	64	528	688
	SW2	116	584	744	132	596	764
	NS2	48	544	692	92	556	716
	KL2	48	516	664	64	524	684
ChainTaskset	SWL	472	940	1172	644	1108	1356
	SWH	628	1100	1336	804	1268	1516
	NSL	468	940	1176	644	1112	1356
	NSH	620	1092	1324	796	1264	1508
GetTasksetRef		52	44	52	52	52	52

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		88	80	88	88	88	88
AssignTaskset		44	40	44	44	44	44
RemoveTaskset		88	88	88	88	88	88
TestSubTaskset		116	108	112	116	116	116
TestEquivalentTaskset		104	100	104	104	104	104
TickSchedule	SW	276	804	952	352	840	1000
	NS	240	768	916	316	800	960
	KL	212	728	876	276	764	924
	SW2	276	808	956	352	820	976
	NS2	236	764	916	312	780	936
	KL2	212	728	876	276	740	900
AdvanceSchedule	SW	224	728	880	280	764	924
	NS	172	692	840	240	724	884
	KL	144	652	796	196	684	844
	SW2	224	728	876	276	744	900
	NS2	172	692	840	236	704	864
	KL2	144	648	800	196	664	820
StartSchedule		168	164	164	168	168	168
StopSchedule		144	144	148	144	144	144
GetScheduleStatus		152	148	152	152	152	152
GetScheduleValue		160	152	156	160	160	160
GetScheduleNext		52	48	56	52	52	52
SetScheduleNext		52	44	52	48	52	48
GetArrivalpointDelay		52	44	52	48	52	48
SetArrivalpointDelay		48	40	48	48	48	48
GetArrivalpointTasksetRef		44	36	40	40	44	40
GetArrivalpointNext		52	44	48	52	52	52
SetArrivalpointNext		48	40	48	48	48	48
TestArrivalpointWritable		96	92	96	92	96	92
GetExecutionTime		20	20	24	24	20	24
GetLargestExecutionTime		48	44	48	48	48	48
ResetLargestExecutionTime		32	32	36	36	32	36
GetStackOffset		72	68	72	72	72	72

Timing

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
Service	Variant						
ActivateTask	SW	168	264	392	176	236	396
	NS	120	232	360	132	208	364
	KL	84	180	308	92	156	312
TerminateTask	LExt	0	0	0	0	0	0
	H	584	576	580	584	580	580
ChainTask	SWL	824	920	1132	1016	1080	1324
	SWH	988	1096	1308	1184	1256	1500
	NSL	824	920	1132	1020	1084	1324
	NSH	980	1084	1296	1172	1244	1488
Schedule	SW	140	140	180	144	136	176
GetTaskID		72	72	76	76	72	72
GetTaskState		124	120	128	152	148	148
EnableAllInterrupts		44	40	48	44	44	44
DisableAllInterrupts		64	60	64	64	60	60
ResumeAllInterrupts		72	68	76	72	72	72
SuspendAllInterrupts		92	88	92	92	88	88
ResumeOSInterrupts		72	68	76	72	72	72
SuspendOSInterrupts		92	88	92	92	88	88
GetResource	Task	80	76	88	80	76	84
	Combined	116	112	112	116	108	108
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	116	112	112	116	108	108
	Combined	156	152	156	156	152	152
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	164	164	168
	NS	n/a	n/a	n/a	168	164	164
	KL	n/a	n/a	n/a	140	136	140
ClearEvent		n/a	n/a	n/a	92	92	88
GetEvent		n/a	n/a	n/a	60	56	56
WaitEvent	<default>	n/a	n/a	n/a	1232	1228	1344
	fp	n/a	n/a	n/a	1256	1252	1368
GetAlarmBase		124	120	124	128	120	120
GetAlarm		144	140	144	144	140	144

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		168	168	168	172	164	168
SetAbsAlarm		188	180	184	184	180	180
CancelAlarm		124	120	124	128	120	120
InitCounter		104	104	104	104	100	104
GetCounterValue		112	108	108	112	104	104
osek_tick_alarm	<default>	164	160	164	164	160	160
	KL	112	104	108	112	104	104
osek_incr_counter		44	40	44	44	40	40
GetActiveApplicationMode		24	20	24	24	20	20
StartOS		2752	2752	2752	2752	2752	2752
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	68	64	68	68	64	64
InitCOM		28	28	28	32	28	28
CloseCOM		28	24	32	32	28	28
StartCOM		80	72	76	372	368	368
StopCOM		52	48	52	56	48	48
ReadFlag		n/a	n/a	n/a	40	36	36
ResetFlag		n/a	n/a	n/a	36	32	32
ReceiveMessage		92	84	88	496	492	496
GetMessageResource		n/a	n/a	n/a	184	180	184
ReleaseMessageResource		n/a	n/a	n/a	208	204	204
GetMessageStatus		n/a	n/a	n/a	136	132	132
SendMessage	SW	300	396	524	732	796	952
	NS	248	360	484	680	756	916
	KL	176	268	400	620	684	840
ActivateTaskset	SW	116	584	744	132	592	764
	NS	48	548	692	92	552	712
	KL	44	516	660	64	520	684
	SW2	116	584	744	128	592	760
	NS2	44	544	692	92	552	712
	KL2	48	512	664	60	524	680
ChainTaskset	SWL	800	1272	1508	988	1448	1696
	SWH	964	1436	1672	1152	1612	1860
	NSL	800	1268	1508	992	1452	1696
	NSH	956	1428	1660	1140	1604	1848
GetTasksetRef		52	44	48	52	48	44

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		88	80	84	88	84	80
AssignTaskset		44	40	44	44	40	40
RemoveTaskset		88	84	92	88	84	88
TestSubTaskset		116	108	112	112	108	108
TestEquivalentTaskset		104	100	104	104	100	100
TickSchedule	SW	280	808	956	352	836	996
	NS	240	764	916	312	800	956
	KL	212	728	876	276	760	920
	SW2	280	808	956	352	812	972
	NS2	240	764	916	316	776	932
	KL2	212	728	876	272	736	896
AdvanceSchedule	SW	224	728	880	280	760	920
	NS	176	692	840	240	720	884
	KL	144	652	800	196	680	840
	SW2	224	728	880	276	740	900
	NS2	176	692	840	236	700	860
	KL2	144	648	796	196	660	820
StartSchedule		164	164	168	164	164	160
StopSchedule		148	140	148	148	140	144
GetScheduleStatus		152	148	152	152	148	148
GetScheduleValue		156	156	156	156	156	152
GetScheduleNext		56	48	52	56	48	52
SetScheduleNext		52	48	48	52	48	48
GetArrivalpointDelay		52	48	48	52	48	48
SetArrivalpointDelay		48	44	44	48	44	44
GetArrivalpointTasksetRef		40	40	40	40	40	36
GetArrivalpointNext		48	48	48	48	48	44
SetArrivalpointNext		48	44	44	48	44	44
TestArrivalpointWritable		96	92	96	96	92	92
GetExecutionTime		176	168	172	176	168	172
GetLargestExecutionTime		64	56	60	64	56	60
ResetLargestExecutionTime		56	52	56	56	52	52
GetStackOffset		72	64	72	72	64	68

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	616	720	840	624	692	848
	NS	700	804	924	704	776	932
	KL	504	616	732	508	584	744
TerminateTask	LExt	660	656	660	652	660	660
	H	816	816	816	812	816	816
ChainTask	SWL	1412	1516	1724	1604	1672	1916
	SWH	1576	1680	1888	1768	1840	2080
	NSL	1516	1620	1824	1708	1776	2020
	NSH	1672	1776	1984	1864	1932	2176
Schedule	SW	276	276	316	272	276	316
GetTaskID		96	96	96	92	96	96
GetTaskState		616	616	616	620	628	624
EnableAllInterrupts		68	68	72	68	72	72
DisableAllInterrupts		96	96	92	92	92	92
ResumeAllInterrupts		120	120	120	116	120	120
SuspendAllInterrupts		124	124	120	116	120	120
ResumeOSInterrupts		120	120	116	116	116	116
SuspendOSInterrupts		120	120	124	116	124	124
GetResource	Task	1080	1080	628	1168	1168	716
	Combined	568	568	568	656	660	660
	CLEx	652	648	648	728	736	736
ReleaseResource	Task	596	596	600	688	688	688
	Combined	584	584	584	668	672	676
	CLEx	588	592	592	672	676	676
SetEvent	SW	n/a	n/a	n/a	684	688	688
	NS	n/a	n/a	n/a	724	728	732
	KL	n/a	n/a	n/a	600	604	612
ClearEvent		n/a	n/a	n/a	180	184	188
GetEvent		n/a	n/a	n/a	520	524	524
WaitEvent	<default>	n/a	n/a	n/a	1460	1464	1560
	fp	n/a	n/a	n/a	1488	1492	1592
GetAlarmBase		472	468	468	464	472	468
GetAlarm		488	488	488	484	488	488

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		580	576	580	576	576	576
SetAbsAlarm		572	572	572	568	572	572
CancelAlarm		452	452	452	448	452	452
InitCounter		448	448	448	448	448	452
GetCounterValue		424	428	424	420	424	424
osek_tick_alarm	<default>	204	204	204	200	200	204
	KL	112	112	112	104	108	112
osek_incr_counter		44	44	44	40	44	44
GetActiveApplicationMode		24	20	24	20	24	24
StartOS		2876	2876	2876	2880	2876	2876
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	84	84	84	80	84	80
InitCOM		28	28	28	28	32	32
CloseCOM		32	28	32	28	32	32
StartCOM		116	120	120	412	416	416
StopCOM		88	88	88	84	88	88
ReadFlag		n/a	n/a	n/a	88	92	92
ResetFlag		n/a	n/a	n/a	84	84	88
ReceiveMessage		376	376	376	768	772	772
GetMessageResource		n/a	n/a	n/a	996	1000	1000
ReleaseMessageResource		n/a	n/a	n/a	976	980	980
GetMessageStatus		n/a	n/a	n/a	336	340	340
SendMessage	SW	1012	1116	1236	1424	1496	1652
	NS	1092	1196	1312	1504	1572	1732
	KL	844	952	1072	1256	1336	1492
ActivateTaskset	SW	880	1404	1548	904	1380	1548
	NS	948	1476	1616	972	1452	1616
	KL	780	1304	1444	804	1280	1448
	SW2	880	1408	1548	904	1380	1548
	NS2	948	1476	1620	976	1452	1620
	KL2	780	1304	1444	800	1280	1448
ChainTaskset	SWL	1732	2256	2484	1924	2396	2656
	SWH	1892	2416	2644	2080	2560	2816
	NSL	1836	2360	2592	2028	2504	2764
	NSH	1988	2512	2744	2176	2652	2912
GetTasksetRef		464	464	464	460	464	464

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		476	476	476	472	476	476
AssignTaskset		372	372	372	372	376	376
RemoveTaskset		476	476	476	476	480	480
TestSubTaskset		516	516	516	512	512	512
TestEquivalentTaskset		508	508	508	500	504	504
TickSchedule	SW	564	1716	1860	1216	1760	1928
	NS	688	1828	1972	1328	1872	2040
	KL	456	1616	1760	1112	1660	1828
	SW2	568	1720	1860	1216	1692	1860
	NS2	688	1832	1972	1328	1804	1976
	KL2	460	1616	1756	1112	1592	1760
AdvanceSchedule	SW	520	1672	1812	1168	1716	1880
	NS	628	1772	1912	1268	1812	1980
	KL	428	1580	1720	1072	1620	1788
	SW2	520	1668	1812	1168	1644	1816
	NS2	628	1772	1912	1268	1744	1912
	KL2	428	1580	1720	1076	1552	1724
StartSchedule		392	392	392	388	392	392
StopSchedule		304	304	304	296	304	300
GetScheduleStatus		336	332	336	332	336	336
GetScheduleValue		308	312	308	308	308	312
GetScheduleNext		156	156	156	148	156	156
SetScheduleNext		324	328	324	320	324	328
GetArrivalpointDelay		224	228	224	224	224	228
SetArrivalpointDelay		268	264	268	264	268	264
GetArrivalpointTasksetRef		192	192	192	188	192	188
GetArrivalpointNext		200	200	200	196	200	200
SetArrivalpointNext		384	380	384	380	384	384
TestArrivalpointWritable		236	236	236	232	236	236
GetExecutionTime		236	236	236	228	236	232
GetLargestExecutionTime		444	444	444	436	444	444
ResetLargestExecutionTime		420	420	420	420	420	420
GetStackOffset		72	72	72	68	72	72

4.3.2 OS Start-up Time

OS start-up time is the time from the entry to the `StartOS()` function to the execution of the first instruction in a user task (including the idle task) without any hook routines being called. This time is always application dependent, since `StartOS()` may activate any number of tasks and start any number of user-specified alarms.

4.3.3 Interrupt Latencies

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. The following tables give the interrupt latencies (in CPU cycles).

Standard

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
Operation	ISR Category						
ISR Latency	Cat 1	68	68	64	68	64	68
	Cat 2	76	76	76	76	76	76

Timing

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
Operation	ISR Category						
ISR Latency	Cat 1	68	68	68	68	68	68
	Cat 2	404	404	404	404	404	404

Extended

Configuration		Application Uses					
	Events	No			Yes		
	Shared Task Priorities	No		Yes	No		Yes
	Multiple Task Activations	No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	68	68	64	68	64	68
	Cat 2	404	404	404	404	404	404

4.3.4 Task Switching Times

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()`).

RTA-OSEK sub-task types also affect the switching time. The tables in this section show the switching times (in CPU cycles) for all system classes for basic, lightweight tasks and for basic and extended heavyweight tasks.

Figures 1 to 8 show the RTA-OSEK switching contexts measured.

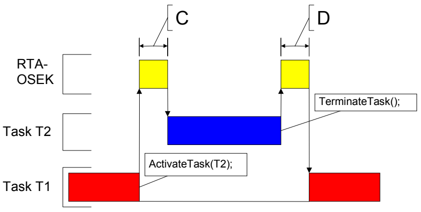

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

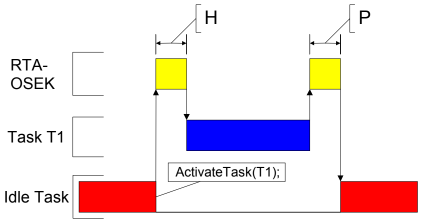

Figure 3: Task Activation from Idle Task

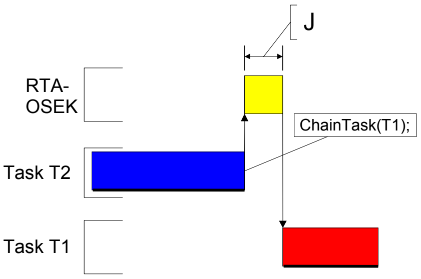

Figure 2: Task Chaining

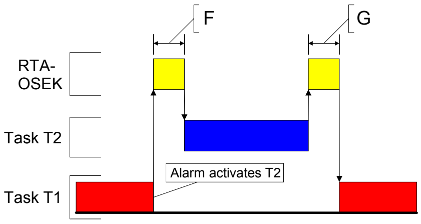

Figure 4: Task Activation from an Alarm

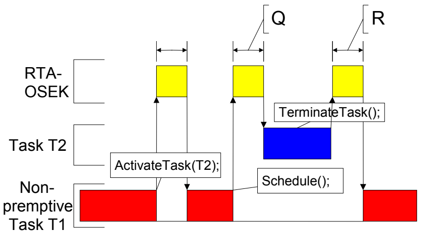

Figure 5: Non-Premptive Task Calls Schedule()

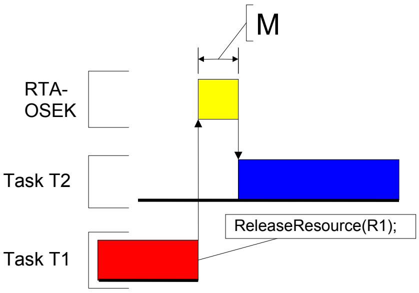

Figure 6: Blocked Task Activated by ReleaseResource()

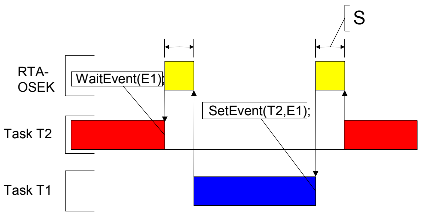

Figure 7: Waiting Task Activated by SetEvent()

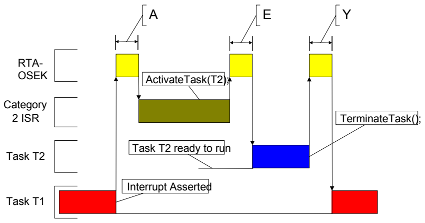

Figure 8: Category 2 ISR Activates a Higher Priority Task

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Normal termination	Light, Basic	132	192	272	132	196	272
Figure 1: D	Heavy, Basic/Extended	292	344	428	372	372	460
ChainTask	Light, Basic	308	452	660	312	452	700
Figure 2: J	Heavy, Basic/Extended	800	1012	1304	892	1040	1372
Pre-emption	Light, Basic	292	440	660	304	444	696
Figure 1: C	Heavy, Basic/Extended	480	576	800	660	728	980
From idle task	Light, Basic	296	440	664	304	444	700
Figure 3: H	Heavy, Basic/Extended	476	580	800	660	728	984
Triggered by alarm	Light, Basic	464	608	832	472	612	868

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Figure 4: F	Heavy, Basic/Extended	652	748	968	832	900	1152
Schedule	Light, Basic	252	296	432	252	300	432
Figure 5: Q	Heavy, Basic/Extended	440	432	568	612	612	744
Release resource	Light, Basic	280	320	420	280	324	420
Figure 6: M	Heavy, Basic/Extended	464	460	560	640	636	736
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	996	992	1292
From category 2 ISR	Light, Basic	316	360	460	316	364	460
Figure 8: E	Heavy, Basic/Extended	504	496	596	676	676	772

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Normal termination	Light, Basic	420	464	552	424	468	548
Figure 1: D	Heavy, Basic/Extended	580	620	704	648	644	732
ChainTask	Light, Basic	676	784	996	684	784	1024
Figure 2: J	Heavy, Basic/Extended	1468	1628	1920	1540	1648	1980
Pre-emption	Light, Basic	552	664	884	564	660	916
Figure 1: C	Heavy, Basic/Extended	700	800	1024	896	956	1212
From idle task	Light, Basic	552	660	884	564	660	916
Figure 3: H	Heavy, Basic/Extended	700	796	1024	896	960	1212
Triggered by alarm	Light, Basic	724	832	1056	736	832	1084

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Figure 4: F	Heavy, Basic/Extended	872	968	1192	1068	1128	1384
Schedule	Light, Basic	512	520	652	512	516	648
Figure 5: Q	Heavy, Basic/Extended	660	656	792	848	840	976
Release resource	Light, Basic	540	544	644	540	544	640
Figure 6: M	Heavy, Basic/Extended	688	680	780	872	868	964
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1200	1196	1508
From category 2 ISR	Light, Basic	788	796	896	788	796	892
Figure 8: E	Heavy, Basic/Extended	936	932	1032	1124	1120	1216

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Normal termination	Light, Basic	660	708	788	652	708	788
Figure 1: D	Heavy, Basic/Extended	816	860	940	876	880	972
ChainTask	Light, Basic	1260	1372	1580	1268	1372	1616
Figure 2: J	Heavy, Basic/Extended	2288	2440	2728	2356	2464	2800
Pre-emption	Light, Basic	960	1072	1288	964	1068	1320
Figure 1: C	Heavy, Basic/Extended	1108	1212	1428	1300	1368	1624
From idle task	Light, Basic	960	1072	1284	964	1068	1320
Figure 3: H	Heavy, Basic/Extended	1108	1212	1428	1300	1368	1624
Triggered by alarm	Light, Basic	1168	1280	1496	1176	1280	1532
Figure 4: F	Heavy,	132	142	163	151	158	183

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
	Basic/Extended	0	0	6	2	0	2
Schedule	Light, Basic	596	608	740	596	608	740
Figure 5: Q	Heavy, Basic/Extended	748	748	880	932	936	1068
Release resource	Light, Basic	948	956	1052	1036	1044	1144
Figure 6: M	Heavy, Basic/Extended	1096	1096	1192	1368	1372	1468
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1692	1692	2000
From category 2 ISR	Light, Basic	848	860	956	848	860	956
Figure 8: E	Heavy, Basic/Extended	1000	1000	1096	1180	1184	1284

4.4 Configuration of Run-time Context

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 are effectively overlaid. The RTA-OSEK GUI is able to calculate the worst-case stack requirement for the entire application, based on the declared stack usage, the priorities and the resource occupation of individual tasks.

The size of the run-time context of a task depends on the task type and the system configuration. The following tables give the sizes (in bytes) for different OS status and configurations:

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		50	50	50	50	50	50
BCC1 lightweight, floating-point		54	54	54	54	54	54
BCC1 heavyweight, integer		70	70	70	70	70	70
BCC1 heavyweight, floating-point		70	70	70	70	70	70

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
BCC2 lightweight, integer	n/a	54	54	n/a	54	54
BCC2 lightweight, floating-point	n/a	54	54	n/a	54	54
BCC2 heavyweight, integer	n/a	70	70	n/a	70	70
BCC2 heavyweight, floating-point	n/a	70	70	n/a	70	70
ECC1 heavyweight, integer	n/a	n/a	n/a	70	70	70
ECC1 heavyweight, floating-point	n/a	n/a	n/a	70	70	70
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	70
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	70
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	50	50	50	50	50	50
BCC1 lightweight, floating-point	54	54	54	54	54	54
BCC1 heavyweight, integer	70	70	70	70	70	70
BCC1 heavyweight, floating-point	70	70	70	70	70	70
BCC2 lightweight, integer	n/a	54	54	n/a	54	54
BCC2 lightweight, floating-point	n/a	54	54	n/a	54	54
BCC2 heavyweight, integer	n/a	70	70	n/a	70	70
BCC2 heavyweight, floating-point	n/a	70	70	n/a	70	70
ECC1 heavyweight, integer	n/a	n/a	n/a	70	70	70
ECC1 heavyweight, floating-point	n/a	n/a	n/a	70	70	70
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	70
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	70

Timing

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Pre- and Post-Task hooks not used						
Task type						
BCC1 lightweight, integer	58	58	58	58	58	58
BCC1 lightweight, floating-point	62	62	62	62	62	62
BCC1 heavyweight, integer	74	74	74	74	74	74

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 heavyweight, floating-point		74	74	74	74	74	74
BCC2 lightweight, integer		n/a	62	62	n/a	62	62
BCC2 lightweight, floating-point		n/a	62	62	n/a	62	62
BCC2 heavyweight, integer		n/a	74	74	n/a	74	74
BCC2 heavyweight, floating-point		n/a	74	74	n/a	74	74
ECC1 heavyweight, integer		n/a	n/a	n/a	76	76	76
ECC1 heavyweight, floating-point		n/a	n/a	n/a	76	76	76
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	76
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	76
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		58	58	58	58	58	58
BCC1 lightweight, floating-point		62	62	62	62	62	62
BCC1 heavyweight, integer		74	74	74	74	74	74
BCC1 heavyweight, floating-point		74	74	74	74	74	74
BCC2 lightweight, integer		n/a	62	62	n/a	62	62
BCC2 lightweight, floating-point		n/a	62	62	n/a	62	62
BCC2 heavyweight, integer		n/a	74	74	n/a	74	74
BCC2 heavyweight, floating-point		n/a	74	74	n/a	74	74
ECC1 heavyweight, integer		n/a	n/a	n/a	76	76	76
ECC1 heavyweight, floating-point		n/a	n/a	n/a	76	76	76
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	76
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	76

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		58	58	58	58	58	58
BCC1 lightweight, floating-point		62	62	62	62	62	62

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 heavyweight, integer		74	74	74	74	74	74
BCC1 heavyweight, floating-point		74	74	74	74	74	74
BCC2 lightweight, integer		n/a	62	62	n/a	62	62
BCC2 lightweight, floating-point		n/a	62	62	n/a	62	62
BCC2 heavyweight, integer		n/a	74	74	n/a	74	74
BCC2 heavyweight, floating-point		n/a	74	74	n/a	74	74
ECC1 heavyweight, integer		n/a	n/a	n/a	78	78	78
ECC1 heavyweight, floating-point		n/a	n/a	n/a	78	78	78
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	78
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	78
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		58	58	58	58	58	58
BCC1 lightweight, floating-point		62	62	62	62	62	62
BCC1 heavyweight, integer		74	74	74	74	74	74
BCC1 heavyweight, floating-point		74	74	74	74	74	74
BCC2 lightweight, integer		n/a	62	62	n/a	62	62
BCC2 lightweight, floating-point		n/a	62	62	n/a	62	62
BCC2 heavyweight, integer		n/a	74	74	n/a	74	74
BCC2 heavyweight, floating-point		n/a	74	74	n/a	74	74
ECC1 heavyweight, integer		n/a	n/a	n/a	78	78	78
ECC1 heavyweight, floating-point		n/a	n/a	n/a	78	78	78
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	78
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	78

5 Interrupt Exit Behavior

Here we illustrate the problem with the C16x interrupt exit behavior, where two cycles of the interrupted code are always executed, even when there is another interrupt already pending.

Consider an application that contains a section of code to perform a memory check. This must execute with interrupts locked out, since memory will be in an inconsistent state during the test. Let us suppose that the duration of the resulting interrupt lock causes interrupts to be delayed unacceptably. A standard solution is to enable interrupts “momentarily” at some point during the test when memory is in a safe, consistent state. This effectively breaks the interrupt lock into two smaller portions, each of which can be tolerated by other interrupts in the system.

That is, we transform this code

```
DisableAllInterrupts();
/* Memory test code section. */
EnableAllInterrupts();
```

into this code:

```
DisableAllInterrupts();
/* Memory test code first section. */
EnableAllInterrupts ();
DisableAllInterrupts();
/* Memory test code second section. */
EnableAllInterrupts ();
```

The intention behind the enabling and disabling of interrupts is that *all* pending interrupts get to run before entering the second section of non-interruptible code. However, on the 167 only a limited number of the highest priority interrupts will actually get to run at that point. Low priority interrupts can be blocked for the sum of the durations of the two sections.

For example, if it takes three cycles between enabling interrupts and disabling them again, then only two interrupt handlers can run in this window; the first gets to run immediately. Interrupts are enabled, but then two cycles of interrupted code are executed before entering the second interrupt handler. On exit from the second interrupt handler, the code to disable interrupts will be executed and no more interrupts will be handled until the second non-interruptible section of code has completed. This is shown in the following figure:

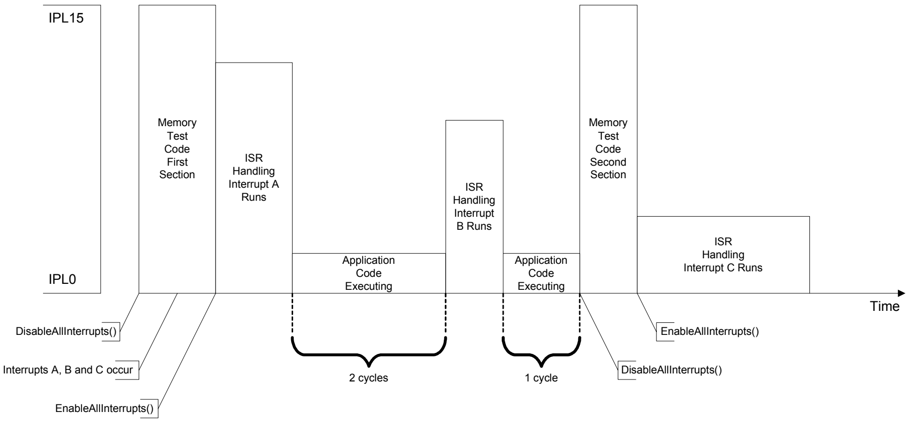
IPL15
IPL0
Memory Test Code First Section
ISR Handling Interrupt A Runs
Application Code Executing
ISR Handling Interrupt B Runs
Application Code Executing
Memory Test Code Second Section
ISR Handling Interrupt C Runs
Time
DisableAllInterrupts()
Interrupts A, B and C occur
EnableAllInterrupts()
2 cycles
1 cycle
EnableAllInterrupts()
DisableAllInterrupts()

Figure 9 : Uncorrected Behavior

Thus, it is possible for an interrupt handler to run later than predicted by any analysis that assumes all pending interrupts run when interrupts are enabled. Borderline systems may run correctly for hundreds of hours in testing, but still prove unreliable in mass manufacture. It is necessary to change this behavior to allow all pending interrupts to be handled when interrupts are enabled.

One way to achieve this in software is to consume the two cycles after a `RETI` instruction before allowing interrupted code to resume, as shown in Figure 10 below. This allows the processor to handle a pending interrupt without executing interrupted code. We suggest this is accomplished by executing the terminal `RETI` twice before returning execution to the interrupted code. This requires the extension of the stack as shown below (assuming that the `RETI` instruction is labeled `_reti`).

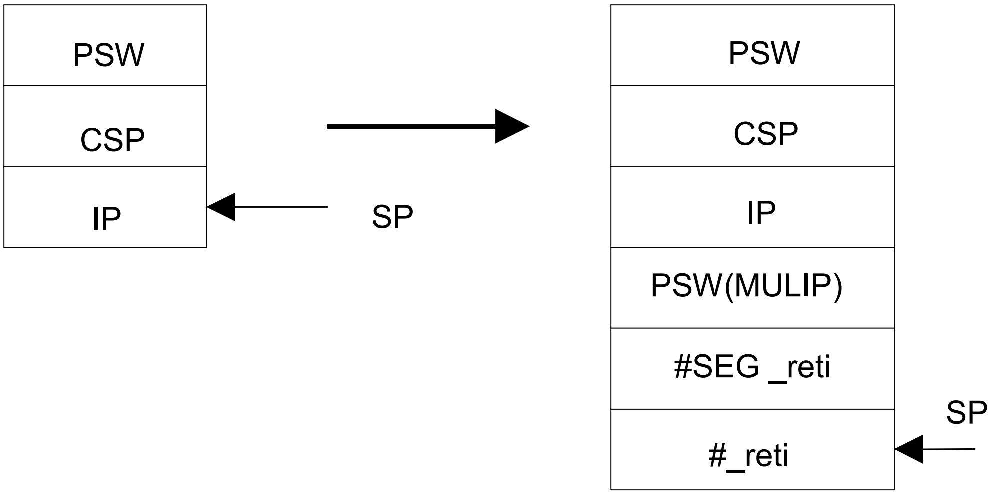

Figure 10: Stack Extension

PSW(MULIP) is the value of the *stacked* PSW with the MULIP bit set to the contents of the *interrupt handler's* MULIP bit. This ensures that the contents of the MULIP bit is not corrupted by the first execution of the `RETI` instruction.

The return address for the interrupt is thus set so that the handler terminates with a RETI instruction that returns to itself and only then returns to the interrupted code. The second execution of the RETI instruction consumes enough execution to effectively prevent control returning to the interrupted code until after the processor has handled a pending interrupt. A corollary of this is that RTA-OSEK may only be used on processor steps (versions) that correctly support interrupts during a RETI instruction. The corrected behavior is illustrated below:

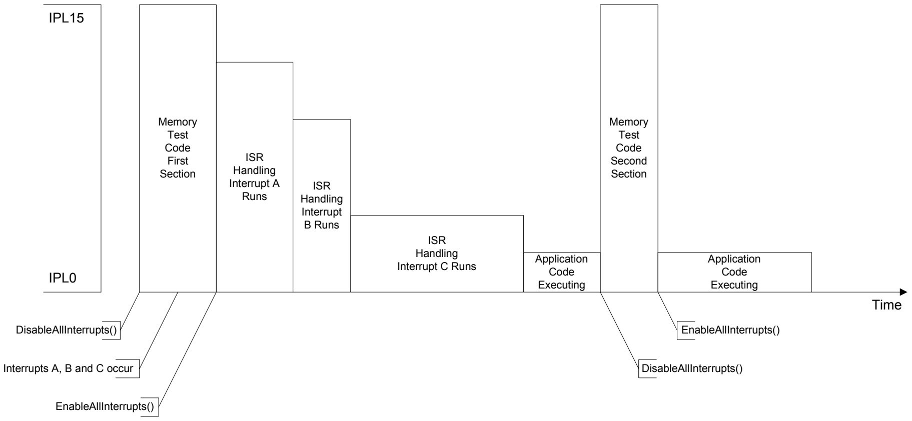
IPL15
IPL0
Memory Test Code First Section
ISR Handling Interrupt A Runs
ISR Handling Interrupt B Runs
ISR Handling Interrupt C Runs
Application Code Executing
Memory Test Code Second Section
Application Code Executing
Time
DisableAllInterrupts()
Interrupts A, B and C occur
EnableAllInterrupts()
EnableAllInterrupts()
DisableAllInterrupts()

Figure 11 : Corrected Behavior

Important: Note that Category 2 interrupt handlers are normal C functions, so they do not terminate with a RETI instruction and do not require such user code at exit. Category 2 interrupt handlers are dispatched from an RTA-OSEK interrupt wrapper that exits as described here.

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.