RTA-OSEK

Binding Manual: Tasking/167

Contact Details

ETAS Group www.etasgroup.com	
ETAS GmbH 70469 Stuttgart, Germany Tel.:+49 711 89661-0 Fax:+49 711 89661-300 sales.de@etas.com	**ETAS Inc.** Ann Arbor, MI 48103, USA Tel.: +1 888 ETAS INC Fax: +1 734 997 9449 sales.us@etas.com
ETAS K.K. Yokohama 220-6217, Japan Tel.: +81 45 222-0900 Fax: +81 45 222-0956 sales.jp@etas.com	**ETAS S.A.S.** 94588 Rungis Cedex, France Tel.: +33 (1) 56 70 00 50 Fax: +33 (1) 56 70 00 51 sales.fr@etas.com
ETAS Korea Co. Ltd. Seoul 137-889, Korea Tel.: +82 2 5747-016 Fax: +82 2 5747-120 sales.kr@etas.com	**ETAS Ltd.** Burton-upon-Trent Staffordshire DE14 2WQ, UK Tel.: +44 1283 54 65 12 Fax: +44 1283 54 87 67 sales.uk@etas.com
ETAS (Shanghai) Co., Ltd. Shanghai 200120, P.R. China Tel.: +86 21 5037 2220 Fax: +86 21 5037 2221 sales.cn@etas.com	**ETAS Automotive India Pvt. Ltd.** Bangalore 560 068, India Tel.: +91 80 4191 2585 Fax: +91 80 4191 2586 sales.in@etas.com

Copyright Notice

Version: B00087-002

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, RTA-TRACE 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 target-specific information for the Tasking/167 port of LiveDevices' RTA-OSEK. It supplements the more general information in the *RTA-OSEK User Guide*.

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?

The reader should have an understanding of real time embedded programming in an OSEK context. 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.

Program code, file names, C types and symbols, 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

This chapter contains important details about RTA-OSEK and your toolchain. A port of the RTA-OSEK Component 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.

If you are interested in using a different version of the same toolchain, please 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.

DPP3 (Data page Pointer Register 3) should be fixed at address 0xC000.

2.2 Compiler

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

Vendor	Tasking
Compiler	Tasking VX toolset for C166/ST10
Version	v2.3r2 Build 592

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

Option	Description
`-Mn`	Near memory model
`-C<cpu>`	Selects target CPU

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

Environment variable CPU_TYPE is needed and should be set up to C16x target variant (e.g. c167cr) irrespective of the target variant to avoid linking problem.

The Tasking supplied C startup code, `cstart.c`, can be used unmodified. It is important that the included header file `cstart.h` has the appropriate settings for the target CPU. This is demonstrated in the example application, which is discussed in more detail in the *RTA-OSEK Getting Started Guide*.

The Tasking VX toolchain is shipped with a driver program called `cc166.exe` which can be used for compiling and linking the application.

Important: There is a conflict between the OSEK defined type state symbol READY and definition in certain Tasking supplied SRF files. This can give rise to compiler warning and information messages when compiling a C file, such as:

Warning: c166 W504: ["<compiler path>\include\ sfr\regxc167ci.sfr" 15452 /9] macro "READY" redefined.

Information: c166 I801: ["<RTA-OSEK install directory>\Task167\inc\oscore.h" 378/9] previous definition of macro "READY"

Take care that the correct definition is used when comparing with the output value of GetTaskState().

2.3 Assembler

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

Vendor	Tasking
Assembler	Tasking VX-toolset for C166: Assembler
Version	v2.3r2 Build 268

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 uninitialized data; initialized during StartOS()
os_pur	RAM	RTA-OSEK initialized data; zeroed during StartOS()
os_text	ROM	RTA-OSEK code
os_data2	RAM	RTA-OSEK data; must be initialized during C-startup
os_trace_ram	RAM	RTA-TRACE uninitialized data; must be zeroed during C-startup

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
cstart.c	Startup code with automatic initilization
pnearp32	C library function
icall	C library function
cpnnw	C library function
vectab	C library function
mul	C library function
_init	C library function
_dcti	C library function
_Exit	C library function

This port of RTA-OSEK is built for the `c166cnlib, c166rtn.lib` runtime libraries. The compiler flag +Mn specifies all functions to be @near (small memory model).

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 C16x Family 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 for Tasking/167. 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 *Infineon XC16x User 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

RTA-OSEK does not impose any restrictions on which interrupt vectors may be used.

The valid base addresses for the vector table are:

Base Address	Notes
0x0	For C16x
0x xx 0000	For XC16x (xx is defined in Reg VECSEG the segment where the vector table is located to)

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, #SOF _reti
  push R1
  ; restore registers, including R1
  …
  atomic #2
_reti: LABEL FAR
  reti                    ; This instruction
                             ; executes twice
```

3.3 Register Settings

The RTA-OSEK Component requires the following registers to be initialized before calling `StartOS()`.

PSW Field	Required Value	Notes
IEN	1	Permits interrupts

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.

`osStackOffsetType` 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 - osStackOffsetType()

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)	28

Timing

Stack	API Max Usage (Bytes)
System stack	8
User stack (C stack)	28

Extended

Stack	API Max Usage (Bytes)
System stack	8
User stack (C stack)	28

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

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Limits for the number of application modes	255					

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	27	27	27	27	27	27
	ROM	111	111	115	191	191	195
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	45	45	45	45	45	45
	ROM	177	177	181	257	257	261
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	53	53	53	53	53	53
	ROM	203	203	207	283	283	287
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	32	36	n/a	32	36
ECC1, Integer task	RAM	n/a	n/a	n/a	28	28	28
	ROM	n/a	n/a	n/a	42	42	42
ECC1, floating-point task	RAM	n/a	n/a	n/a	30	30	30
	ROM	n/a	n/a	n/a	42	42	42
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	46
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	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	46	46	46	46	46	46
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	10	10	10	10	10	10
	ROM	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	104	104	104	104	104	104
Message	RAM	11	11	11	31	31	31
	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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		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
ScheduleTable	RAM	10	10	10	10	10	10
	ROM	82	82	82	82	82	82
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	8	8	8	8	8	8
Arrivalpoint (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8
Schedule	RAM	12	12	12	12	12	12
	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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	10	10	10	10	10	10
	ROM	32	32	32	32	32	32
BCC1 Heavyweight task	RAM	12	12	12	12	12	12
	ROM	34	34	34	34	34	34
BCC2 task	RAM	n/a	14	16	n/a	14	16
	ROM	n/a	40	44	n/a	40	44
ECC1, Integer task	RAM	n/a	n/a	n/a	38	38	38
	ROM	n/a	n/a	n/a	50	50	50
ECC1, floating-point task	RAM	n/a	n/a	n/a	40	40	40
	ROM	n/a	n/a	n/a	50	50	50
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	40

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	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	42
	ROM	n/a	n/a	n/a	n/a	n/a	54
Category 2 ISR	RAM	10	10	10	10	10	10
	ROM	60	60	60	60	60	60
Category 2 ISR, floating-point	RAM	12	12	12	12	12	12
	ROM	68	68	68	68	68	68
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	10	10	10	10	10	10
	ROM	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	104	104	104	104	104	104
Message	RAM	11	11	11	31	31	31
	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
ScheduleTable	RAM	10	10	10	10	10	10
	ROM	82	82	82	82	82	82
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	8	8	8	8	8	8
Arrivalpoint (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8
Schedule	RAM	12	12	12	12	12	12

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	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	12	12	12	12	12	12
	ROM	36	36	36	36	36	36
BCC1 Heavyweight task	RAM	14	14	14	14	14	14
	ROM	36	36	36	36	36	36
BCC2 task	RAM	n/a	16	18	n/a	16	18
	ROM	n/a	42	46	n/a	42	46
ECC1, Integer task	RAM	n/a	n/a	n/a	40	40	40
	ROM	n/a	n/a	n/a	52	52	52
ECC1, floating-point task	RAM	n/a	n/a	n/a	42	42	42
	ROM	n/a	n/a	n/a	52	52	52
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	42
	ROM	n/a	n/a	n/a	n/a	n/a	56
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	44
	ROM	n/a	n/a	n/a	n/a	n/a	56
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	64	64	64	64	64	64
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	72	72	72	72	72	72
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Alarm	RAM	10	10	10	10	10	10
	ROM	32	32	32	32	32	32
Counter	RAM	4	4	4	4	4	4
	ROM	106	106	106	106	106	106
Message	RAM	11	11	11	31	31	31
	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
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
ScheduleTable	RAM	10	10	10	10	10	10
	ROM	82	82	82	82	82	82
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	14	14	14	14	14	14
Arrivalpoint (writable)	RAM	14	14	14	14	14	14
	ROM	14	14	14	14	14	14
Schedule	RAM	16	16	16	16	16	16
	ROM	32	32	32	32	32	32
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.
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.

Variant	Description
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	236	356	456	244	372	496
	NS		196	316	416	204	324	456
	KL	2	124	244	344	132	252	384
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	32	32	32	32	32	32
ChainTask	SWL	1, 8	188	308	408	196	316	448
	SWH	1, 9	228	344	456	248	356	496
	NSL	8	188	308	408	196	316	448
	NSH	9	220	336	448	240	348	488
Schedule			148	148	212	148	148	212
GetTaskID			48	48	48	48	48	48
GetTaskState			120	120	120	148	148	148
EnableAllInterrupts			36	36	36	36	36	36
DisableAllInterrupts			44	44	44	44	44	44
ResumeAllInterrupts			58	58	58	58	58	58
SuspendAllInterrupts			64	64	64	64	64	64
ResumeOSInterrupts			58	58	58	58	58	58
SuspendOSInterrupts			94	94	94	94	94	94
GetResource	Task	7	40	40	52	40	40	52
	Combined	6	130	130	130	130	130	130
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	120	120	120	120	120	120
	Combined	6	272	272	272	272	272	272
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	216	216	376
	NS		n/a	n/a	n/a	168	168	328
	NS1i	10	n/a	n/a	n/a	88	n/a	n/a
	KL	2	n/a	n/a	n/a	116	116	276
	KL1i	2, 10	n/a	n/a	n/a	36	n/a	n/a
ClearEvent			n/a	n/a	n/a	76	76	76

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetEvent			n/a	n/a	n/a	28	28	28
WaitEvent	<default>		n/a	n/a	n/a	516	516	980
	fp	11	n/a	n/a	n/a	592	592	1140
	1i	10	n/a	n/a	n/a	36	n/a	n/a
GetAlarmBase			92	92	92	92	92	92
GetAlarm			232	232	232	232	232	232
SetRelAlarm			1524	1524	1524	1524	1524	1524
SetAbsAlarm			1892	1892	1892	1892	1892	1892
CancelAlarm			172	172	172	172	172	172
InitCounter			136	136	136	136	136	136
GetCounterValue			172	172	172	172	172	172
GetScheduleTableStatus		34	78	96	96	78	96	96
NextScheduleTable		34	100	196	196	100	196	196
StartScheduleTable		34	126	176	176	126	176	176
StopScheduleTable		34	86	104	104	86	104	104
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	10	10	10
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		28	28	28	28	28	28
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	24	24	24	24	24	24
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		160	160	160	160	160	160
	KL	2	108	108	108	108	108	108
osek_incr_counter			128	128	128	128	128	128
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			196	196	196	196	196	196
ShutdownOS	NoHook	12	62	62	62	62	62	62
	Hook	13	70	70	70	70	70	70
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			40	40	40	40	40	40
StopCOM			28	28	28	28	28	28
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	112	112	112	380	380	380

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	380	380	380	380	380	380
GetMessageResource			64	64	64	64	64	64
ReleaseMessageResource			52	52	52	52	52	52
GetMessageStatus			100	100	100	100	100	100
SendMessage	SW CCCA	1, 14	200	200	200	492	492	492
	SW CCCB	1, 15	464	464	464	492	492	492
	NS CCCA	14	200	200	200	492	492	492
	NS CCCB	15	464	464	464	492	492	492
	KL CCCA	2, 14	148	148	148	440	440	440
	KL CCCB	2, 15	412	412	412	440	440	440
main_dispatch	NoHook	12	232	232	304	232	232	304
	Hook	13	300	300	368	300	300	368
sub_dispatch	B1LF	19	64	64	64	64	64	64
	B1HI	20	168	168	168	168	168	168
	B1HF	21	184	184	184	184	184	184
	B2LI	22	n/a	136	200	n/a	136	200
	B2LF	23	n/a	152	216	n/a	152	216
	B2HI	24	n/a	462	652	n/a	462	652
	B2HF	25	n/a	478	668	n/a	478	668
	E1HI	26	n/a	n/a	n/a	614	614	782
	E1HF	27	n/a	n/a	n/a	630	630	798
	E2HI	28	n/a	n/a	n/a	n/a	n/a	782
	E2HF	29	n/a	n/a	n/a	n/a	n/a	798
ErrorHook support		16	52	52	52	52	52	52
	ServiceID	17	64	64	64	64	64	64
	Parameters	18	116	116	116	116	116	116
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	232	420	540	252	460	624
	NS		192	380	492	212	420	576
	KL	2	120	308	420	140	348	504
ChainTaskset	SWL	1, 8	196	408	520	208	436	576
	SWH	1, 9	244	472	588	256	500	656
	NSL	8	196	408	520	208	436	576
	NSH	9	236	464	580	248	492	648
GetTasksetRef			12	12	12	12	12	12

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
MergeTaskset			72	72	72	72	72	72
AssignTaskset			12	12	12	12	12	12
RemoveTaskset			76	76	76	76	76	76
TestSubTaskset			92	92	92	92	92	92
TestEquivalentTaskset			84	84	84	84	84	84
TickSchedule	SW	1	460	412	412	412	412	412
	NS		412	348	348	348	348	348
	KL	2	352	288	288	288	288	288
AdvanceSchedule	SW	1	428	352	352	352	352	352
	NS		380	304	304	304	304	304
	KL	2	328	244	244	244	244	244
StartSchedule			180	180	180	180	180	180
StopSchedule			136	136	136	136	136	136
GetScheduleStatus			212	212	212	212	212	212
GetScheduleValue			188	188	188	188	188	188
GetScheduleNext			16	16	16	16	16	16
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			80	80	80	80	80	80
SetArrivalpointDelay			76	76	76	76	76	76
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			20	20	20	20	20	20
SetArrivalpointNext			16	16	16	16	16	16
TestArrivalpointWritable			48	48	48	48	48	48
GetExecutionTime			12	12	12	12	12	12
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			8	8	8	8	8	8
GetStackOffset			56	56	56	56	56	56

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	236	356	456	244	372	496
	NS		196	316	416	204	324	456
	KL	2	124	244	344	132	252	384
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	32	32	32	32	32	32
ChainTask	SWL	1, 8	188	308	408	196	316	448
	SWH	1, 9	228	344	456	248	356	496
	NSL	8	188	308	408	196	316	448
	NSH	9	220	336	448	240	348	488
Schedule			180	180	244	180	180	244
GetTaskID			48	48	48	48	48	48
GetTaskState			120	120	120	148	148	148
EnableAllInterrupts			36	36	36	36	36	36
DisableAllInterrupts			44	44	44	44	44	44
ResumeAllInterrupts			58	58	58	58	58	58
SuspendAllInterrupts			64	64	64	64	64	64
ResumeOSInterrupts			58	58	58	58	58	58
SuspendOSInterrupts			94	94	94	94	94	94
GetResource	Task	7	40	40	52	40	40	52
	Combined	6	130	130	130	130	130	130
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	152	152	152	152	152	152
	Combined	6	336	336	336	336	336	336
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	216	216	376
	NS		n/a	n/a	n/a	168	168	328
	NS1i	10	n/a	n/a	n/a	88	n/a	n/a
	KL	2	n/a	n/a	n/a	116	116	276
	KL1i	2, 10	n/a	n/a	n/a	36	n/a	n/a
ClearEvent			n/a	n/a	n/a	76	76	76
GetEvent			n/a	n/a	n/a	28	28	28
WaitEvent	<default>		n/a	n/a	n/a	688	688	1148
	fp	11	n/a	n/a	n/a	756	756	1304

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	172	n/a	n/a
GetAlarmBase			92	92	92	92	92	92
GetAlarm			232	232	232	232	232	232
SetRelAlarm			1524	1524	1524	1524	1524	1524
SetAbsAlarm			1892	1892	1892	1892	1892	1892
CancelAlarm			172	172	172	172	172	172
InitCounter			136	136	136	136	136	136
GetCounterValue			172	172	172	172	172	172
GetScheduleTableStatus		34	78	96	96	78	96	96
NextScheduleTable		34	100	196	196	100	196	196
StartScheduleTable		34	126	176	176	126	176	176
StopScheduleTable		34	86	104	104	86	104	104
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	10	10	10
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		28	28	28	28	28	28
GetISRID		4	28	28	28	28	28	28
Process container	Yielding	32	24	24	24	24	24	24
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		160	160	160	160	160	160
	KL	2	108	108	108	108	108	108
osek_incr_counter			128	128	128	128	128	128
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			312	312	312	312	312	312
ShutdownOS	NoHook	12	62	62	62	62	62	62
	Hook	13	70	70	70	70	70	70
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			40	40	40	40	40	40
StopCOM			28	28	28	28	28	28
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	112	112	112	380	380	380
	CCCB	15	380	380	380	380	380	380
GetMessageResource			64	64	64	64	64	64
ReleaseMessageResource			52	52	52	52	52	52

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			100	100	100	100	100	100
SendMessage	SW CCCA	1, 14	200	200	200	492	492	492
	SW CCCB	1, 15	464	464	464	492	492	492
	NS CCCA	14	200	200	200	492	492	492
	NS CCCB	15	464	464	464	492	492	492
	KL CCCA	2, 14	148	148	148	440	440	440
	KL CCCB	2, 15	412	412	412	440	440	440
main_dispatch	NoHook	12	388	388	456	388	388	456
	Hook	13	448	448	524	448	448	524
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	144	144	144	144	144	144
	B1HF	21	160	160	160	160	160	160
	B2LI	22	n/a	76	140	n/a	76	140
	B2LF	23	n/a	88	152	n/a	88	152
	B2HI	24	n/a	400	588	n/a	400	588
	B2HF	25	n/a	416	604	n/a	416	604
	E1HI	26	n/a	n/a	n/a	636	636	804
	E1HF	27	n/a	n/a	n/a	652	652	820
	E2HI	28	n/a	n/a	n/a	n/a	n/a	804
	E2HF	29	n/a	n/a	n/a	n/a	n/a	820
ErrorHook support		16	52	52	52	52	52	52
	ServiceID	17	64	64	64	64	64	64
	Parameters	18	116	116	116	116	116	116
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	152	152	152	152	152	152
Timing_termination		4	180	180	180	180	180	180
ActivateTaskset	SW	1	232	420	540	252	460	624
	NS		192	380	492	212	420	576
	KL	2	120	308	420	140	348	504
ChainTaskset	SWL	1, 8	196	408	520	208	436	576
	SWH	1, 9	244	472	588	256	500	656
	NSL	8	196	408	520	208	436	576
	NSH	9	236	464	580	248	492	648
GetTasksetRef			12	12	12	12	12	12
MergeTaskset			72	72	72	72	72	72
AssignTaskset			12	12	12	12	12	12
RemoveTaskset			76	76	76	76	76	76

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			92	92	92	92	92	92
TestEquivalentTaskset			84	84	84	84	84	84
TickSchedule	SW	1	460	412	412	412	412	412
	NS		412	348	348	348	348	348
	KL	2	352	288	288	288	288	288
AdvanceSchedule	SW	1	428	352	352	352	352	352
	NS		380	304	304	304	304	304
	KL	2	328	244	244	244	244	244
StartSchedule			180	180	180	180	180	180
StopSchedule			136	136	136	136	136	136
GetScheduleStatus			212	212	212	212	212	212
GetScheduleValue			188	188	188	188	188	188
GetScheduleNext			16	16	16	16	16	16
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			80	80	80	80	80	80
SetArrivalpointDelay			76	76	76	76	76	76
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			20	20	20	20	20	20
SetArrivalpointNext			16	16	16	16	16	16
TestArrivalpointWritable			48	48	48	48	48	48
GetExecutionTime			196	196	196	196	196	196
GetLargestExecutionTime			88	88	88	88	88	88
ResetLargestExecutionTime			88	88	88	88	88	88
GetStackOffset			56	56	56	56	56	56

Extended

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	436	556	656	456	564	700
	NS		500	624	724	520	632	764
	KL	2	288	408	508	308	416	552
TerminateTask	LExt	3	200	200	200	200	200	200

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	256	256	256	256	256	256
ChainTask	SWL	1, 8	472	604	704	492	612	744
	SWH	1, 9	544	672	772	564	680	812
	NSL	8	556	688	788	580	700	828
	NSH	9	628	756	856	652	768	896
Schedule			424	424	488	424	424	488
GetTaskID			72	72	72	72	72	72
GetTaskState			452	452	452	452	452	452
EnableAllInterrupts			60	60	60	60	60	60
DisableAllInterrupts			76	76	76	76	76	76
ResumeAllInterrupts			172	172	172	172	172	172
SuspendAllInterrupts			96	96	96	96	96	96
ResumeOSInterrupts			172	172	172	172	172	172
SuspendOSInterrupts			126	126	126	126	126	126
GetResource	Task	7	652	652	612	652	652	612
	Combined	6	642	642	642	642	642	642
	CLEx	3	592	592	592	592	592	592
ReleaseResource	Task	7	600	600	600	600	600	600
	Combined	6	790	790	790	790	790	790
	CLEx	3	552	552	552	552	552	552
SetEvent	SW	1	n/a	n/a	n/a	560	560	724
	NS		n/a	n/a	n/a	628	628	792
	NS1i	10	n/a	n/a	n/a	440	n/a	n/a
	KL	2	n/a	n/a	n/a	412	412	576
	KL1i	2, 10	n/a	n/a	n/a	328	n/a	n/a
ClearEvent			n/a	n/a	n/a	252	252	252
GetEvent			n/a	n/a	n/a	336	336	336
WaitEvent	<default>		n/a	n/a	n/a	936	936	1376
	fp	11	n/a	n/a	n/a	1004	1004	1532
	1i	10	n/a	n/a	n/a	408	n/a	n/a
GetAlarmBase			376	376	376	376	376	376
GetAlarm			380	380	380	380	380	380
SetRelAlarm			1804	1804	1804	1804	1804	1804
SetAbsAlarm			2160	2160	2160	2160	2160	2160
CancelAlarm			316	316	316	316	316	316
InitCounter			500	500	500	500	500	500
GetCounterValue			472	472	472	472	472	472

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	94	112	112	94	112	112
NextScheduleTable		34	116	212	212	116	212	212
StartScheduleTable		34	142	192	192	142	192	192
StopScheduleTable		34	102	120	120	102	120	120
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	10	10	10
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		28	28	28	28	28	28
GetISRID		4	44	44	44	44	44	44
Process container	Yielding	32	24	24	24	24	24	24
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		268	268	268	268	268	268
	KL	2	108	108	108	108	108	108
osek_incr_counter			128	128	128	128	128	128
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			344	344	344	344	344	344
ShutdownOS	NoHook	12	78	78	78	78	78	78
	Hook	13	86	86	86	86	86	86
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			72	72	72	72	72	72
StopCOM			72	72	72	72	72	72
ReadFlag			48	48	48	48	48	48
ResetFlag			56	56	56	56	56	56
ReceiveMessage	CCCA	14	308	308	308	572	572	572
	CCCB	15	572	572	572	572	572	572
GetMessageResource			156	156	156	156	156	156
ReleaseMessageResource			156	156	156	156	156	156
GetMessageStatus			192	192	192	192	192	192
SendMessage	SW CCCA	1, 14	392	392	392	672	672	672
	SW CCCB	1, 15	644	644	644	672	672	672
	NS CCCA	14	392	392	392	672	672	672
	NS CCCB	15	644	644	644	672	672	672
	KL CCCA	2, 14	300	300	300	580	580	580
	KL CCCB	2, 15	552	552	552	580	580	580
main_dispatch	NoHook	12	388	388	456	388	388	456

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	448	448	524	448	448	524
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	148	148	148	148	148	148
	B1HF	21	164	164	164	164	164	164
	B2LI	22	n/a	76	140	n/a	76	140
	B2LF	23	n/a	88	152	n/a	88	152
	B2HI	24	n/a	404	592	n/a	404	592
	B2HF	25	n/a	420	608	n/a	420	608
	E1HI	26	n/a	n/a	n/a	644	644	812
	E1HF	27	n/a	n/a	n/a	660	660	828
	E2HI	28	n/a	n/a	n/a	n/a	n/a	812
	E2HF	29	n/a	n/a	n/a	n/a	n/a	828
ErrorHook support		16	192	192	192	192	192	192
	ServiceID	17	208	208	208	208	208	208
	Parameters	18	260	260	260	260	260	260
validity_checks		3	40	40	40	40	40	40
Timing_dispatch		4	152	152	152	152	152	152
Timing_termination		4	180	180	180	180	180	180
ActivateTaskset	SW	1	532	688	796	604	740	860
	NS		600	752	864	668	796	928
	KL	2	372	528	640	440	572	708
ChainTaskset	SWL	1, 8	592	746	858	648	770	902
	SWH	1, 9	684	836	952	740	860	1004
	NSL	8	684	862	974	760	886	1018
	NSH	9	768	928	1048	828	956	1092
GetTasksetRef			280	280	280	280	280	280
MergeTaskset			568	568	568	568	568	568
AssignTaskset			388	388	388	388	388	388
RemoveTaskset			572	572	572	572	572	572
TestSubTaskset			592	592	592	592	592	592
TestEquivalentTaskset			580	580	580	580	580	580
TickSchedule	SW	1	720	644	644	644	644	644
	NS		784	792	792	792	792	792
	KL	2	608	532	532	532	532	532
AdvanceSchedule	SW	1	740	636	636	636	636	636
	NS		812	808	808	808	808	808
	KL	2	636	524	524	524	524	524

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			528	528	528	528	528	528
StopSchedule			400	400	400	400	400	400
GetScheduleStatus			516	516	516	516	516	516
GetScheduleValue			504	504	504	504	504	504
GetScheduleNext			216	216	216	216	216	216
SetScheduleNext			380	380	380	380	380	380
GetArrivalpointDelay			440	440	440	440	440	440
SetArrivalpointDelay			468	468	468	468	468	468
GetArrivalpointTasksetRef			288	288	288	288	288	288
GetArrivalpointNext			296	296	296	296	296	296
SetArrivalpointNext			436	436	436	436	436	436
TestArrivalpointWritable			324	324	324	324	324	324
GetExecutionTime			284	284	284	284	284	284
GetLargestExecutionTime			400	400	400	400	400	400
ResetLargestExecutionTime			336	336	336	336	336	336
GetStackOffset			56	56	56	56	56	56

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
32	Container for 2 process functions, not highest priority
33	Container for 2 process functions, highest or APPMODE or ISR
34	code varies with number of schedule tables; example uses 2 schedule tables

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	79	125	171	85	115	177
	NS	65	111	159	71	99	161
	KL	39	85	129	43	71	135
TerminateTask	LExt	0	0	0	0	0	0
	H	153	153	153	155	153	155
ChainTask	SWL	229	283	363	331	357	455
	SWH	319	369	455	423	447	549
	NSL	229	281	365	329	359	455
	NSH	317	365	451	419	445	545
Schedule	SW	71	69	83	73	69	83
GetTaskID		27	27	27	29	27	29
GetTaskState		57	59	59	71	71	73
EnableAllInterrupts		19	19	21	21	19	21
DisableAllInterrupts		23	23	23	25	23	23
ResumeAllInterrupts		27	27	27	29	27	29
SuspendAllInterrupts		31	31	31	31	29	31
ResumeOSInterrupts		27	27	27	29	27	29
SuspendOSInterrupts		29	29	29	33	31	33
GetResource	Task	31	31	35	33	31	39
	Combined	49	49	51	51	51	51
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	57	57	57	59	57	59
	Combined	79	79	79	81	79	81
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	95	93	91
	NS	n/a	n/a	n/a	79	77	75
	KL	n/a	n/a	n/a	59	57	55
ClearEvent		n/a	n/a	n/a	41	39	41
GetEvent		n/a	n/a	n/a	25	23	25
WaitEvent	<default>	n/a	n/a	n/a	449	447	497

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	fp	n/a	n/a	n/a	463	461	505
GetAlarmBase		101	101	101	105	101	103
GetAlarm		77	77	79	81	79	79
SetRelAlarm		181	181	181	183	181	183
SetAbsAlarm		185	185	185	185	185	187
CancelAlarm		63	63	63	65	63	65
InitCounter		63	63	63	65	63	65
GetCounterValue		71	71	69	73	71	73
osek_tick_alarm	<default>	73	73	73	77	73	75
	KL	51	51	51	51	49	51
osek_incr_counter		17	17	17	19	19	19
GetActiveApplicationMode		7	7	7	7	5	7
StartOS		423	423	423	423	423	425
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	33	33	33	35	33	35
InitCOM		9	9	9	11	9	11
CloseCOM		9	9	9	11	9	11
StartCOM		33	33	33	149	147	149
StopCOM		17	17	15	19	15	19
ReadFlag		n/a	n/a	n/a	13	9	13
ResetFlag		n/a	n/a	n/a	11	9	11
ReceiveMessage		43	43	43	191	189	191
GetMessageResource		n/a	n/a	n/a	77	75	77
ReleaseMessageResource		n/a	n/a	n/a	99	97	97
GetMessageStatus		n/a	n/a	n/a	45	43	47
SendMessage	SW	165	213	259	317	345	407
	NS	155	201	247	303	329	393
	KL	105	149	197	251	279	343
ActivateTaskset	SW	63	297	359	71	299	375
	NS	51	283	337	59	287	353
	KL	25	255	311	29	259	327
	SW2	65	295	357	71	301	373
	NS2	51	283	339	57	287	355
	KL2	23	257	311	31	261	327
ChainTaskset	SWL	221	461	553	319	549	651
	SWH	311	551	641	407	639	741

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NSL	223	461	553	319	549	651
	NSH	307	549	639	405	635	739
GetTasksetRef		17	17	17	19	17	21
MergeTaskset		41	41	41	43	41	43
AssignTaskset		17	17	17	19	17	19
RemoveTaskset		43	43	43	45	43	43
TestSubTaskset		51	51	51	53	51	51
TestEquivalentTaskset		47	47	47	49	47	49
TickSchedule	SW	173	435	491	209	447	511
	NS	153	407	463	181	419	485
	KL	125	379	437	153	391	457
	SW2	173	435	491	209	439	507
	NS2	153	407	463	181	411	479
	KL2	125	381	435	155	383	453
AdvanceSchedule	SW	137	403	459	177	415	479
	NS	117	383	439	155	395	459
	KL	99	357	411	129	367	433
	SW2	139	403	457	177	407	475
	NS2	117	383	439	157	387	455
	KL2	97	357	411	131	359	427
StartSchedule		97	97	97	99	97	101
StopSchedule		87	87	87	89	87	89
GetScheduleStatus		93	93	93	95	93	95
GetScheduleValue		99	99	99	101	99	101
GetScheduleNext		21	19	19	23	19	21
SetScheduleNext		19	19	19	21	21	21
GetArrivalpointDelay		47	47	47	49	47	47
SetArrivalpointDelay		45	45	45	47	45	47
GetArrivalpointTasksetRef		17	17	17	19	15	19
GetArrivalpointNext		19	19	19	21	19	23
SetArrivalpointNext		19	17	17	21	17	21
TestArrivalpointWritable		31	29	29	33	29	33
GetExecutionTime		9	11	11	13	11	13
GetLargestExecutionTime		23	25	25	27	25	25
ResetLargestExecutionTime		11	11	11	13	11	11
GetStackOffset		33	33	33	35	33	35

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	79	125	173	83	115	177
	NS	67	111	161	69	101	163
	KL	39	83	133	41	75	135
TerminateTask	LExt	0	0	0	0	0	0
	H	315	315	319	317	319	319
ChainTask	SWL	437	489	573	543	575	665
	SWH	513	561	649	621	649	743
	NSL	437	491	573	543	575	669
	NSH	509	555	645	619	645	741
Schedule	SW	71	71	83	71	71	85
GetTaskID		27	27	29	27	29	29
GetTaskState		59	59	61	71	73	73
EnableAllInterrupts		19	19	23	19	21	21
DisableAllInterrupts		23	21	25	23	25	23
ResumeAllInterrupts		27	27	29	27	29	19
SuspendAllInterrupts		29	29	33	29	31	17
ResumeOSInterrupts		27	27	29	27	29	19
SuspendOSInterrupts		31	31	31	31	33	19
GetResource	Task	31	33	37	31	33	37
	Combined	51	49	53	51	51	53
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	57	57	59	57	59	59
	Combined	79	81	81	79	81	81
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	95	97	91
	NS	n/a	n/a	n/a	79	79	75
	KL	n/a	n/a	n/a	57	59	55
ClearEvent		n/a	n/a	n/a	39	41	41
GetEvent		n/a	n/a	n/a	23	25	23
WaitEvent	<default>	n/a	n/a	n/a	631	633	675
	fp	n/a	n/a	n/a	641	643	687
GetAlarmBase		103	103	103	103	103	103
GetAlarm		79	79	81	79	81	79

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		181	181	183	181	183	183
SetAbsAlarm		185	185	187	185	185	187
CancelAlarm		63	63	65	63	65	65
InitCounter		63	63	65	63	65	65
GetCounterValue		71	71	71	71	73	71
osek_tick_alarm	<default>	73	73	75	73	77	75
	KL	51	51	51	49	51	51
osek_incr_counter		17	17	19	19	21	21
GetActiveApplicationMode		7	5	7	7	9	9
StartOS		1321	1321	1321	1321	1321	1321
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	33	33	35	33	35	35
InitCOM		9	9	11	9	11	11
CloseCOM		9	9	11	9	11	11
StartCOM		33	33	35	147	149	149
StopCOM		17	17	19	17	19	19
ReadFlag		n/a	n/a	n/a	9	13	13
ResetFlag		n/a	n/a	n/a	9	11	11
ReceiveMessage		45	43	45	189	191	191
GetMessageResource		n/a	n/a	n/a	75	77	77
ReleaseMessageResource		n/a	n/a	n/a	95	97	97
GetMessageStatus		n/a	n/a	n/a	45	45	45
SendMessage	SW	165	211	261	313	347	407
	NS	157	201	249	299	331	395
	KL	105	149	199	249	281	343
ActivateTaskset	SW	65	295	359	69	301	375
	NS	51	283	339	55	289	355
	KL	23	257	313	29	261	327
	SW2	65	295	359	69	303	373
	NS2	51	283	339	55	287	353
	KL2	25	255	313	29	261	327
ChainTaskset	SWL	431	669	763	533	765	863
	SWH	505	745	835	603	841	939
	NSL	431	669	761	531	765	863
	NSH	501	741	833	603	839	937
GetTasksetRef		17	17	21	19	19	19

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		41	41	43	41	43	41
AssignTaskset		17	17	17	17	19	19
RemoveTaskset		43	43	43	41	45	45
TestSubTaskset		51	51	51	49	53	51
TestEquivalentTaskset		47	47	47	47	49	49
TickSchedule	SW	173	435	493	207	451	511
	NS	153	407	465	179	421	485
	KL	125	381	437	153	393	457
	SW2	173	435	493	207	441	507
	NS2	153	407	465	179	413	479
	KL2	127	381	437	151	385	453
AdvanceSchedule	SW	139	403	461	173	415	479
	NS	119	383	441	155	397	461
	KL	97	357	413	127	369	433
	SW2	137	401	459	173	409	475
	NS2	119	383	441	153	389	455
	KL2	97	357	413	127	363	429
StartSchedule		97	97	99	99	101	99
StopSchedule		87	87	89	87	89	89
GetScheduleStatus		93	93	95	93	95	95
GetScheduleValue		99	99	101	99	101	101
GetScheduleNext		19	19	21	19	21	21
SetScheduleNext		21	19	21	19	21	21
GetArrivalpointDelay		47	47	49	45	47	49
SetArrivalpointDelay		45	45	47	45	47	47
GetArrivalpointTasksetRef		15	17	19	17	19	19
GetArrivalpointNext		19	19	21	21	23	21
SetArrivalpointNext		17	17	19	19	21	19
TestArrivalpointWritable		29	29	31	31	33	31
GetExecutionTime		99	99	101	99	101	101
GetLargestExecutionTime		53	55	57	55	55	57
ResetLargestExecutionTime		45	45	47	45	47	47
GetStackOffset		33	33	35	33	35	35

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	315	357	403	321	347	405
	NS	341	389	437	347	373	431
	KL	269	311	359	277	303	359
TerminateTask	LExt	347	345	347	345	347	345
	H	417	415	417	413	417	413
ChainTask	SWL	745	799	883	851	879	971
	SWH	817	869	951	925	951	1043
	NSL	781	837	919	891	917	1011
	NSH	849	903	985	957	985	1079
Schedule	SW	127	125	139	125	127	137
GetTaskID		37	35	39	35	39	35
GetTaskState		327	323	327	327	329	327
EnableAllInterrupts		31	29	29	29	31	29
DisableAllInterrupts		35	33	37	35	37	35
ResumeAllInterrupts		47	45	47	45	47	45
SuspendAllInterrupts		45	43	43	41	45	41
ResumeOSInterrupts		47	45	47	45	47	45
SuspendOSInterrupts		43	41	45	43	43	43
GetResource	Task	493	491	285	531	533	321
	Combined	263	261	263	301	303	301
	CLEx	289	289	291	325	327	327
ReleaseResource	Task	255	253	255	293	295	293
	Combined	261	259	263	299	301	299
	CLEx	253	251	253	289	291	289
SetEvent	SW	n/a	n/a	n/a	355	357	355
	NS	n/a	n/a	n/a	365	367	363
	KL	n/a	n/a	n/a	313	315	313
ClearEvent		n/a	n/a	n/a	81	83	81
GetEvent		n/a	n/a	n/a	281	283	279
WaitEvent	<default>	n/a	n/a	n/a	727	729	761
	fp	n/a	n/a	n/a	735	737	775
GetAlarmBase		259	257	273	257	259	269
GetAlarm		227	227	243	227	229	241

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		373	371	387	371	371	385
SetAbsAlarm		365	361	377	363	363	375
CancelAlarm		207	205	221	205	207	219
InitCounter		379	375	399	377	377	397
GetCounterValue		209	207	209	207	209	207
osek_tick_alarm	<default>	129	127	127	127	129	127
	KL	51	51	51	49	51	51
osek_incr_counter		21	17	19	19	21	19
GetActiveApplicationMode		7	7	9	5	9	7
StartOS		1357	1357	1357	1357	1357	1357
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	41	39	41	39	41	39
InitCOM		11	9	11	9	11	9
CloseCOM		11	9	11	9	11	9
StartCOM		53	51	53	165	167	165
StopCOM		33	31	33	31	35	33
ReadFlag		n/a	n/a	n/a	31	33	33
ResetFlag		n/a	n/a	n/a	31	33	33
ReceiveMessage		177	177	177	319	323	319
GetMessageResource		n/a	n/a	n/a	441	443	441
ReleaseMessageResource		n/a	n/a	n/a	423	425	423
GetMessageStatus		n/a	n/a	n/a	137	137	137
SendMessage	SW	515	555	605	657	683	741
	NS	543	591	641	683	709	769
	KL	445	485	533	585	609	667
ActivateTaskset	SW	383	647	703	397	653	715
	NS	409	669	725	419	669	737
	KL	333	599	655	345	599	671
	SW2	383	647	703	399	653	715
	NS2	407	669	725	421	671	737
	KL2	333	599	655	345	599	671
ChainTaskset	SWL	833	1109	1201	943	1193	1297
	SWH	909	1175	1267	1019	1259	1361
	NSL	863	1157	1245	987	1241	1341
	NSH	937	1205	1295	1047	1289	1389
GetTasksetRef		257	255	257	255	257	255

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		183	181	181	181	181	181
AssignTaskset		119	117	119	117	119	117
RemoveTaskset		183	181	185	181	183	181
TestSubTaskset		187	185	189	185	187	185
TestEquivalentTaskset		183	181	183	181	183	181
TickSchedule	SW	247	811	869	555	833	903
	NS	271	843	899	587	865	935
	KL	203	767	825	511	789	859
	SW2	245	809	869	555	809	883
	NS2	273	843	899	587	841	915
	KL2	203	767	823	511	767	839
AdvanceSchedule	SW	221	783	841	529	807	877
	NS	253	815	871	559	837	909
	KL	185	741	797	483	763	833
	SW2	221	785	841	527	783	857
	NS2	251	815	871	559	815	887
	KL2	183	741	797	485	739	813
StartSchedule		193	191	191	191	193	191
StopSchedule		139	137	141	137	141	137
GetScheduleStatus		153	153	153	153	153	153
GetScheduleValue		151	149	151	149	149	149
GetScheduleNext		55	53	53	53	55	53
SetScheduleNext		109	107	109	107	107	107
GetArrivalpointDelay		143	139	141	139	141	139
SetArrivalpointDelay		167	165	167	165	167	165
GetArrivalpointTasksetRef		63	61	63	61	65	61
GetArrivalpointNext		65	65	67	65	67	65
SetArrivalpointNext		127	125	125	125	127	125
TestArrivalpointWritable		77	75	77	75	77	75
GetExecutionTime		131	129	131	129	131	129
GetLargestExecutionTime		293	291	293	291	293	291
ResetLargestExecutionTime		277	275	277	275	277	275
GetStackOffset		35	33	35	33	35	33

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	60	60	60	60	60	60
	Cat 2	172	204	204	212	204	212

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	60	60	60	60	60	60
	Cat 2	436	460	460	460	460	460

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	60	60	60	60	60	60
	Cat 2	436	460	460	460	460	460

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.

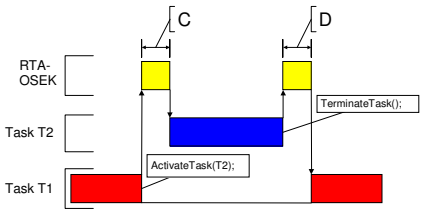

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

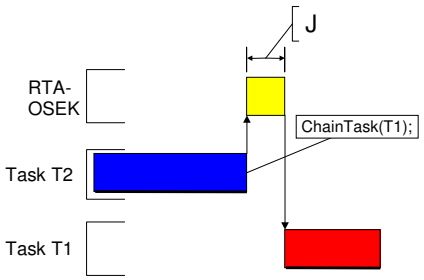

Figure 2: Task Chaining

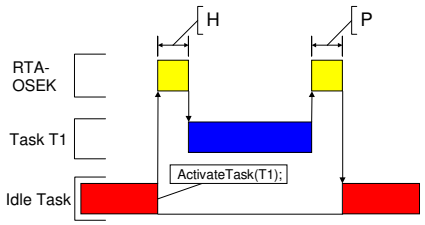

Figure 3: Task Activation from Idle Task

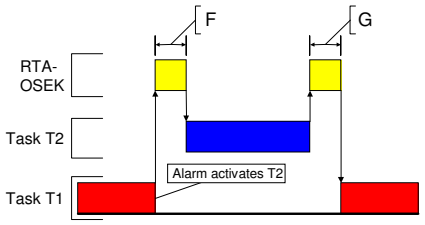

Figure 4: Task Activation from an Alarm

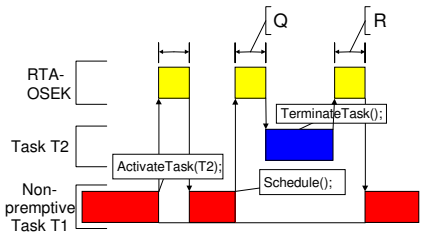

Figure 5: Non-Premptive Task Calls Schedule()

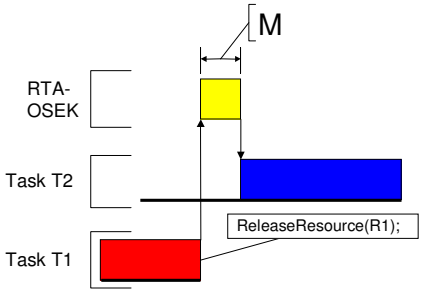

Figure 6: Blocked Task Activated by ReleaseResource()

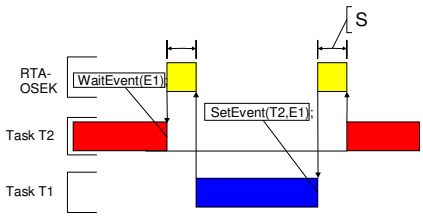

Figure 7: Waiting Task Activated by SetEvent()

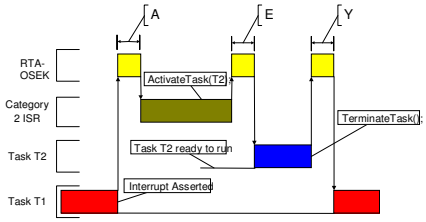

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	67	101	131	69	101	133
Figure 1: D	Heavy, Basic/Extended	153	181	209	191	189	219
ChainTask	Light, Basic	135	207	291	141	209	307
Figure 2: J	Heavy, Basic/Extended	399	495	609	443	505	635
Pre-emption	Light, Basic	143	215	301	147	215	321
Figure 1: C	Heavy, Basic/Extended	239	291	379	339	369	471
From idle task	Light, Basic	143	215	301	147	215	321
Figure 3: H	Heavy, Basic/Extended	239	291	379	341	369	473
Triggered by alarm	Light, Basic	231	305	391	237	307	409
Figure 4: F	Heavy, Basic/Extended	329	381	467	429	459	561
Schedule	Light, Basic	121	147	199	121	147	203
Figure 5: Q	Heavy, Basic/Extended	217	225	277	313	313	367
Release resource	Light, Basic	131	159	199	133	159	203
Figure 6: M	Heavy, Basic/Extended	227	235	277	325	323	367
SetEvent							

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	473	469	597
From category 2 ISR	Light, Basic	111	149	191	125	149	191
Figure 8: E	Heavy, Basic/Extended	207	225	267	317	315	357

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	245	265	293	243	265	295
Figure 1: D	Heavy, Basic/Extended	315	333	363	341	345	371
ChainTask	Light, Basic	367	417	501	371	419	513
Figure 2: J	Heavy, Basic/Extended	781	845	959	809	853	979
Pre-emption	Light, Basic	289	339	425	291	343	443
Figure 1: C	Heavy, Basic/Extended	363	415	503	467	501	601
From idle task	Light, Basic	291	339	425	293	341	443
Figure 3: H	Heavy, Basic/Extended	363	415	503	469	503	601
Triggered by alarm	Light, Basic	381	429	517	383	433	533
Figure 4: F	Heavy, Basic/Extended	453	505	593	559	593	691
Schedule	Light, Basic	267	271	323	267	273	325
Figure 5: Q	Heavy, Basic/Extended	339	347	401	443	443	497
Release resource	Light, Basic	279	283	325	279	285	325
Figure 6: M	Heavy, Basic/Extended	353	361	401	455	457	497
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	587	587	703
From category 2 ISR	Light, Basic	429	447	487	441	449	487
Figure 8: E	Heavy, Basic/Extended	503	523	563	617	621	659

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	347	363	395	345	367	393
Figure 1: D	Heavy, Basic/Extended	415	431	461	441	443	467
ChainTask	Light, Basic	671	725	807	679	727	819
Figure 2: J	Heavy, Basic/Extended	1181	1247	1359	1213	1257	1381
Pre-emption	Light, Basic	509	551	639	513	553	653
Figure 1: C	Heavy, Basic/Extended	581	631	717	691	717	813
From idle task	Light, Basic	507	551	639	513	555	653
Figure 3: H	Heavy, Basic/Extended	583	631	717	691	715	813
Triggered by alarm	Light, Basic	651	693	781	657	697	795
Figure 4: F	Heavy, Basic/Extended	725	771	859	831	857	955
Schedule	Light, Basic	307	309	361	305	311	361
Figure 5: Q	Heavy, Basic/Extended	381	387	441	481	483	531
Release resource	Light, Basic	453	455	497	491	497	535
Figure 6: M	Heavy, Basic/Extended	527	533	575	667	669	707
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	841	843	957
From category 2 ISR	Light, Basic	461	477	517	471	479	515
Figure 8: E	Heavy, Basic/Extended	537	555	595	649	651	687

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	58	58	58	58	58	58
BCC1 heavyweight, integer	80	80	80	80	80	80
BCC1 heavyweight, floating-point	80	80	80	80	80	80
BCC2 lightweight, integer	n/a	58	58	n/a	58	58
BCC2 lightweight, floating-point	n/a	58	58	n/a	58	58
BCC2 heavyweight, integer	n/a	80	80	n/a	80	80
BCC2 heavyweight, floating-point	n/a	80	80	n/a	80	80
ECC1 heavyweight, integer	n/a	n/a	n/a	82	82	82
ECC1 heavyweight, floating-point	n/a	n/a	n/a	82	82	82
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	82
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	82
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	52	52	52	52	52	52
BCC1 lightweight, floating-point	60	60	60	60	60	60
BCC1 heavyweight, integer	82	82	82	82	82	82
BCC1 heavyweight, floating-point	82	82	82	82	82	82
BCC2 lightweight, integer	n/a	60	60	n/a	60	60
BCC2 lightweight, floating-point	n/a	60	60	n/a	60	60
BCC2 heavyweight, integer	n/a	82	82	n/a	82	82
BCC2 heavyweight, floating-point	n/a	82	82	n/a	82	82
ECC1 heavyweight, integer	n/a	n/a	n/a	84	84	84
ECC1 heavyweight, floating-point	n/a	n/a	n/a	84	84	84
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	84
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	84

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		62	62	62	62	62	62
BCC1 lightweight, floating-point		66	66	66	66	66	66
BCC1 heavyweight, integer		88	88	88	88	88	88
BCC1 heavyweight, floating-point		88	88	88	88	88	88
BCC2 lightweight, integer		n/a	66	66	n/a	66	66
BCC2 lightweight, floating-point		n/a	66	66	n/a	66	66
BCC2 heavyweight, integer		n/a	88	88	n/a	88	88
BCC2 heavyweight, floating-point		n/a	88	88	n/a	88	88
ECC1 heavyweight, integer		n/a	n/a	n/a	92	92	92
ECC1 heavyweight, floating-point		n/a	n/a	n/a	92	92	92
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	92
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	92
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		62	62	64	62	62	64
BCC1 lightweight, floating-point		66	66	68	66	66	68
BCC1 heavyweight, integer		88	88	90	88	88	90
BCC1 heavyweight, floating-point		88	88	90	88	88	90
BCC2 lightweight, integer		n/a	66	68	n/a	66	68
BCC2 lightweight, floating-point		n/a	66	68	n/a	66	68
BCC2 heavyweight, integer		n/a	88	90	n/a	88	90
BCC2 heavyweight, floating-point		n/a	88	90	n/a	88	90
ECC1 heavyweight, integer		n/a	n/a	n/a	92	92	94
ECC1 heavyweight, floating-point		n/a	n/a	n/a	92	92	94
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	94
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	94

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	62	62	62	62	62	62
BCC1 lightweight, floating-point	66	66	66	66	66	66
BCC1 heavyweight, integer	88	88	88	88	88	88
BCC1 heavyweight, floating-point	88	88	88	88	88	88
BCC2 lightweight, integer	n/a	66	66	n/a	66	66
BCC2 lightweight, floating-point	n/a	66	66	n/a	66	66
BCC2 heavyweight, integer	n/a	88	88	n/a	88	88
BCC2 heavyweight, floating-point	n/a	88	88	n/a	88	88
ECC1 heavyweight, integer	n/a	n/a	n/a	94	94	94
ECC1 heavyweight, floating-point	n/a	n/a	n/a	94	94	94
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	94
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	94
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	62	62	64	62	62	64
BCC1 lightweight, floating-point	66	66	68	66	66	68
BCC1 heavyweight, integer	88	88	90	88	88	90
BCC1 heavyweight, floating-point	88	88	90	88	88	90
BCC2 lightweight, integer	n/a	66	68	n/a	66	68
BCC2 lightweight, floating-point	n/a	66	68	n/a	66	68
BCC2 heavyweight, integer	n/a	88	90	n/a	88	90
BCC2 heavyweight, floating-point	n/a	88	90	n/a	88	90
ECC1 heavyweight, integer	n/a	n/a	n/a	94	94	96
ECC1 heavyweight, floating-point	n/a	n/a	n/a	94	94	96
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	96
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	96

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:

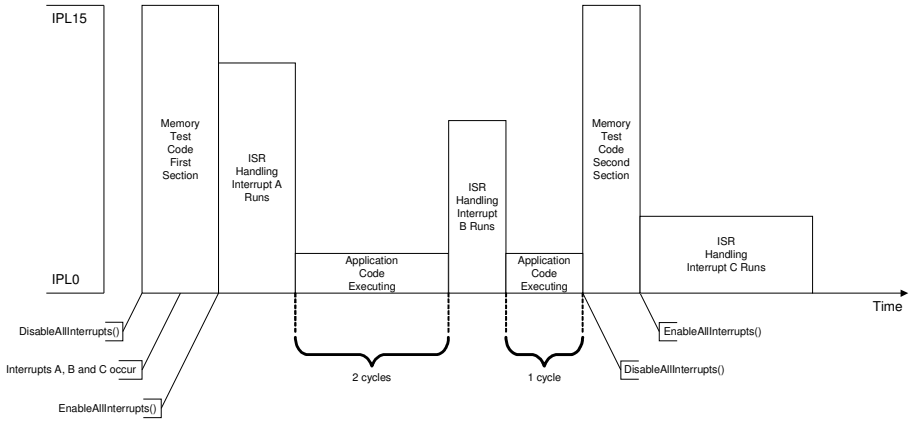

Figure 9 : Uncorrected Behavior

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`).

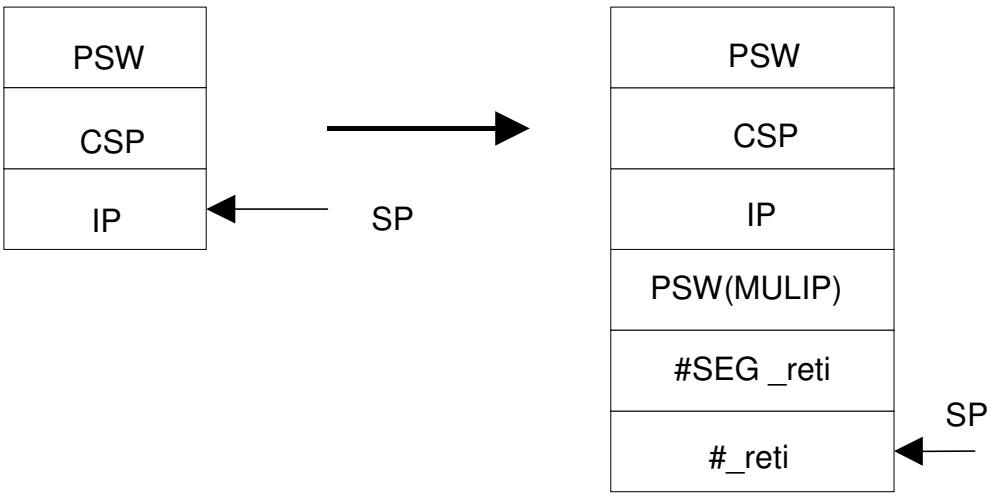

Figure 10: Stack Extension

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:

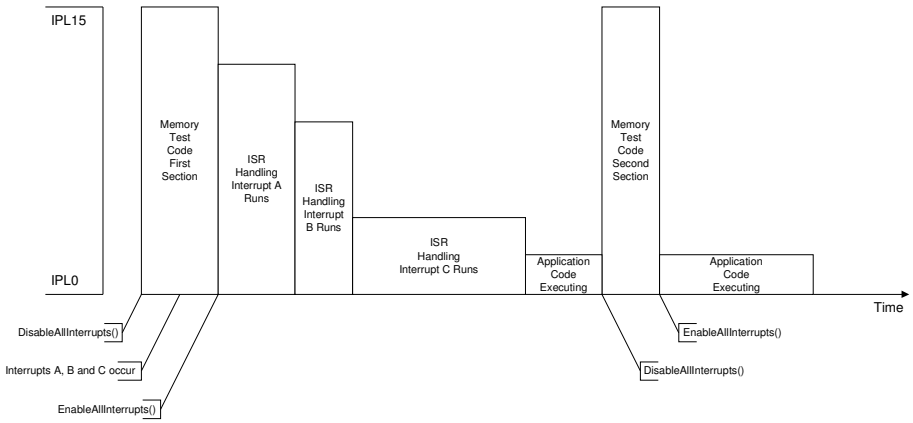

Figure 11 : Corrected Behavior

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.

6 Compatibility with Pre-v5 Kernels

6.1 Updating the Application Version

To convert an existing v3.x OIL configuration file to v5.0x (i.e. v5.00 or v5.01), load the file into the v5 RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.0x. When the OIL configuration file is saved it will then use the v5.0x format and the v5.0x kernel libraries. This process can be reversed to move back to earlier kernel versions.

6.2 32 Bit Timer Drivers

The v3.x kernel uses sixteen bit timer values, whereas the v5.0x kernel uses thirty-two bit timer values. Therefore any existing applications' timer drivers will need modifying. Since the C16X Timer Module provides only sixteen bit timer registers the upper sixteen bits will need to be emulated in software. The provided example application demonstrates one method of achieving this for the `T0` timer register.

6.3 User Stack Pointer

Tasking toolset for C Compiler Version 7.5 release 1 uses Register R0 as User Stack Pointer whereas TASKING VX-toolset for C166 C compiler, and the version v2.3r2 used to build the v5.0 kernel, uses R15 as user stack pointer.

6.4 Variant Support

The v5.0 DLL now directly supports the following variants: ST10, C167, XC161 and XC22XX.

6.5 Data Initialization

Data initialized by compiler Startup code and StartOS() API are now contained in different sections. The new section os_data2 has been created to hold variables that are initialized by the C startup. The existing sections os_pur and os_pir now only hold variables initialized by the StartOS API call.

7 Compatibility with V5 Kernels

7.1 Updating the Application Version

To convert an existing v5.00 OIL configuration file to v5.01, load the file into the v5 RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.01. When the OIL configuration file is saved it will then use v5.01 kernel libraries. This process can be reversed to move back to earlier kernel versions.

Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found at the front of this manual and on the ETAS Group website www.etasgroup.com.