RTA-OSEK

Binding Manual: HC(S)12/Metrowerks

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

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

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

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

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

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: RM00022-002

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 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 HC(S)12/Metrowerks 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 HC(S)12/Metrowerks supports a special zero-page addressing mode, but RTA-OSEK makes no special use of zero-page.

The use of bank-switching for program code is supported within the following guidelines:

1. The RTA-OSEK library must appear in an unswitched bank and must be called with the `__near` calling convention.
2. Application functions which are called by RTA-OSEK must be called with the `__near` calling convention and therefore must appear in unswitched banks. This applies, for example, to ISR entry functions, task entry functions and callback functions.
3. Application functions which use the RTA-OSEK API can be called with the `__far` calling convention and can be placed in switched banks, subject to points 1 and 2 above.

The compiler options `-Mb` and `-Ml` cannot be used on any module which contains functions which call, or are called by, RTA-OSEK. These options cause all functions definitions and all function calls to use the `__far` calling convention, so making use of them will conflict with either point 1 or point 2 above.

2.2 Compiler

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

Vendor	Metrowerks
Compiler	CodeWarrior C Compiler
Version	5.0.24.0

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

Option	Description
`-Cni`	Do not widen char parameters to integers.

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

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

Option	Description
`-Cni`	Do not widen char parameters to integers.
`-NoDebugInfo`	Do not generate debugging information.

2.3 Assembler

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

Vendor	Metrowerks
Assembler	CodeWarrior HC(S)12 Assembler
Version	5.0.26.0

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

2.4 Linker/Locator

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

Option	Description
`-FE`	Only ELF/DWARF format supported
`-WmsgSw1401`	Allow mixture of memory models in compilations

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_intvec`	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
`_COPY`	Structure copying routine

This port of the RTA-OSEK Component is built to run with or without the use of memory bank switching for code and preserves the value in the PPAGE register when necessary. Since the location of the PPAGE register can vary, it

must be made known at linking time, as the value of the label `os_ppage`. This must be done whether or not code bank switching is actually being used.

A recommended method involves assembling a one-byte module (as in the example program) from a file `osppage.asm`, which creates the external label `os_ppage`, in the section `ppage_section`. Then within the PLACEMENT block the linker parameter file must locate `ppage_section` into the address of PPAGE as a one-byte segment. On parts without memory expansion capabilities, there is no PPAGE register and any unused RAM address may be used.

It is also essential to ensure the inclusion of the vector table generated by the configuration tool, which appears in `osgen.asm`. This can be done by including `os_intvex` as one of the ENTRIES and locating `os_intvec` into high memory beginning at `0xFF80`.

If ORTI debugging is to be used, it is also necessary to make sure that the linker retains the address constants generated to support it. This can be done by including `osekdefs.o:*` as one of the ENTRIES.

The linker parameter file used by the example program distributed with RTA-OSEK incorporates all the above requirements.

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	Noral Flex v4.2

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 *CPU12 Reference Manual*.

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

IPL Value	I Bit In Condition Code Register	Description
0	0	User level
1	1	Category 1 and 2 interrupts

3.1.2 Interrupt Vectors

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

Vector	Legality
0xFFFE	RESET cannot be used as Category 1 interrupt.
0xFFF4-0xFFFC	Can be used as restricted Category 1 interrupts only - see section 3.1.5.
0xFF80-0xFFF2	Category 1 or Category 2 interrupts.

The valid base addresses for the vector table are:

Base Address	Notes
0xFF80	This is the only possible base address for the vector table.

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 Metrowerks 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.asm`.

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. 0xFF90	`_os_wrapper_FF90`

The HC(S)12 architecture allows for up to 64 interrupt sources. The six highest priority of these of these are non-maskable, namely three sources of reset, unimplemented opcode trap, SWI and XIRQ. These six are not permitted to be used as OSEK Category 2 interrupts. If they are used as Category 1 interrupts they are subject to the extra restriction that no API calls are permitted within their service routines. Note also that schedulability analysis will not be valid if these interrupt sources are used.

3.2 Register Settings

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

The RTA-OSEK Component does not reserve the use of any hardware registers.

3.3 Stack Usage

3.3.1 Number of Stacks

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

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

3.3.2 Stack Usage within API Calls

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

Standard

API max usage (bytes): 16

Timing

API max usage (bytes): 17

Extended

API max usage (bytes): 26

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.

3.3.3 Initialization of the stack pointer

The RTA-OSEK Component requires the application's startup code to initialize the stack pointer to point to the topmost byte of the SSTACK segment. This occurs automatically unless contrary steps are explicitly taken in the application's linker parameter file.

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	**MC9S12DP256**
Clock speed (MHz)	8
Code memory	On-chip FLASH
Read-only data memory	On-chip FLASH
Read-write data memory	On-chip 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					

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	89	89	89	89	89	89
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	124	124	124	124	124	124
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	28	28	28	28	28	28
	ROM	144	144	144	144	144	144
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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	17	17	17	17	17	17
BCC1 Heavyweight task	RAM	2	2	2	2	2	2
	ROM	19	19	19	19	19	19
BCC2 task	RAM	n/a	3	5	n/a	3	5
	ROM	n/a	20	24	n/a	20	24
ECC1, Integer task	RAM	n/a	n/a	n/a	11	11	11
	ROM	n/a	n/a	n/a	29	29	29
ECC1, floating-point task	RAM	n/a	n/a	n/a	12	12	12
	ROM	n/a	n/a	n/a	29	29	29
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	13
	ROM	n/a	n/a	n/a	n/a	n/a	33
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	14
	ROM	n/a	n/a	n/a	n/a	n/a	33
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	23	23	23	23	23	23
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	32	32	32	32	32	32
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	5	5	5	5	5	5
	ROM	34	34	34	34	34	34
Counter	RAM	2	2	2	2	2	2
	ROM	10	10	10	10	10	10
Message	RAM	11	11	11	51	51	51
	ROM	10	10	10	27	27	27
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	5	0	5	5
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	7	7	7	7	7	7
	ROM	16	16	16	16	16	16
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	5	5	5	5	5	5
	ROM	22	22	22	22	22	22
BCC1 Heavyweight task	RAM	7	7	7	7	7	7
	ROM	24	24	24	24	24	24
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	25	29	n/a	25	29
ECC1, Integer task	RAM	n/a	n/a	n/a	16	16	16
	ROM	n/a	n/a	n/a	34	34	34
ECC1, floating-point task	RAM	n/a	n/a	n/a	17	17	17
	ROM	n/a	n/a	n/a	34	34	34
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	18
	ROM	n/a	n/a	n/a	n/a	n/a	38
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	19
	ROM	n/a	n/a	n/a	n/a	n/a	38

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	5	5	5	5	5	5
	ROM	37	37	37	37	37	37
Category 2 ISR, floating-point	RAM	6	6	6	6	6	6
	ROM	43	43	43	43	43	43
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	5	5	5	5	5	5
	ROM	34	34	34	34	34	34
Counter	RAM	2	2	2	2	2	2
	ROM	10	10	10	10	10	10
Message	RAM	11	11	11	51	51	51
	ROM	10	10	10	27	27	27
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	5	0	5	5
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	7	7	7	7	7	7
	ROM	16	16	16	16	16	16
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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	6	6	6	6	6	6
	ROM	26	26	26	26	26	26
BCC1 Heavyweight task	RAM	8	8	8	8	8	8
	ROM	26	26	26	26	26	26
BCC2 task	RAM	n/a	9	11	n/a	9	11
	ROM	n/a	27	31	n/a	27	31
ECC1, Integer task	RAM	n/a	n/a	n/a	17	17	17
	ROM	n/a	n/a	n/a	36	36	36
ECC1, floating-point task	RAM	n/a	n/a	n/a	18	18	18
	ROM	n/a	n/a	n/a	36	36	36
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	19
	ROM	n/a	n/a	n/a	n/a	n/a	40
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	20
	ROM	n/a	n/a	n/a	n/a	n/a	40
Category 2 ISR	RAM	6	6	6	6	6	6
	ROM	41	41	41	41	41	41
Category 2 ISR, floating-point	RAM	7	7	7	7	7	7
	ROM	47	47	47	47	47	47
Resource	RAM	3	3	3	3	3	3
	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	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Alarm	RAM	5	5	5	5	5	5
	ROM	36	36	36	36	36	36
Counter	RAM	2	2	2	2	2	2
	ROM	12	12	12	12	12	12
Message	RAM	11	11	11	51	51	51
	ROM	12	12	12	29	29	29
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14

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	5	0	5	5
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	9	9	9	9	9	9
	ROM	20	20	20	20	20	20
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	76	125	161	82	131	185
	NS		64	113	149	70	119	173
	KL	2	42	104	140	48	110	162

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	67	116	153	73	122	177
	SWH	1, 9	90	137	173	96	143	198
	NSL	8	67	116	153	73	122	177
	NSH	9	84	131	167	90	137	192
Schedule			51	51	68	51	51	68
GetTaskID			22	22	22	22	22	22
GetTaskState			55	55	55	71	71	71
EnableAllInterrupts			6	6	6	6	6	6
DisableAllInterrupts			13	13	13	13	13	13
ResumeAllInterrupts			15	15	15	15	15	15
SuspendAllInterrupts			23	23	23	23	23	23
ResumeOSInterrupts			15	15	15	15	15	15
SuspendOSInterrupts			29	29	29	29	29	29
GetResource	Task	7	21	21	27	21	21	27
	Combined	6	54	54	54	54	54	54
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	41	41	41	41	41	41
	Combined	6	73	73	73	73	73	73
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	85	85	159
	NS		n/a	n/a	n/a	73	73	147
	NS1i	10	n/a	n/a	n/a	31	n/a	n/a
	KL	2	n/a	n/a	n/a	63	63	137
	KL1i	2, 10	n/a	n/a	n/a	13	n/a	n/a
ClearEvent			n/a	n/a	n/a	19	19	19
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	161	161	313
	fp	11	n/a	n/a	n/a	183	183	361
	1i	10	n/a	n/a	n/a	17	n/a	n/a
GetAlarmBase			32	32	32	32	32	32
GetAlarm			67	67	67	67	67	67
SetRelAlarm			87	87	87	87	87	87
SetAbsAlarm			104	104	104	104	104	104
CancelAlarm			49	49	49	49	49	49
InitCounter			37	37	37	37	37	37

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			43	43	43	43	43	43
osek_tick_alarm	<default>		49	49	49	49	49	49
	KL	2	29	29	29	29	29	29
osek_incr_counter			29	29	29	29	29	29
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			90	90	90	90	90	90
ShutdownOS	NoHook	12	11	11	11	11	11	11
	Hook	13	17	17	17	17	17	17
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			12	12	12	12	12	12
StopCOM			9	9	9	9	9	9
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	40	40	40	123	123	123
	CCCB	15	123	123	123	123	123	123
GetMessageResource			20	20	20	20	20	20
ReleaseMessageResource			18	18	18	18	18	18
GetMessageStatus			37	37	37	37	37	37
SendMessage	SW CCCA	1, 14	59	59	59	162	162	162
	SW CCCB	1, 15	144	144	144	162	162	162
	NS CCCA	14	59	59	59	162	162	162
	NS CCCB	15	144	144	144	162	162	162
	KL CCCA	2, 14	38	38	38	164	164	164
	KL CCCB	2, 15	151	151	151	164	164	164
main_dispatch	NoHook	12	77	77	108	77	77	108
	Hook	13	99	99	130	99	99	130
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	54	54	54	54	54	54
	B1HF	21	60	60	60	60	60	60
	B2LI	22	n/a	38	67	n/a	38	67
	B2LF	23	n/a	43	73	n/a	43	73
	B2HI	24	n/a	85	147	n/a	85	147
	B2HF	25	n/a	91	153	n/a	91	153
	E1HI	26	n/a	n/a	n/a	228	228	296
	E1HF	27	n/a	n/a	n/a	234	234	302
	E2HI	28	n/a	n/a	n/a	n/a	n/a	296

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
	E2HF	29	n/a	n/a	n/a	n/a	n/a	302
ErrorHook support		16	17	17	17	17	17	17
	ServiceID	17	22	22	22	22	22	22
	Parameters	18	37	37	37	37	37	37
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	42	85	134	50	104	154
	NS		30	73	122	38	92	142
	KL	2	14	65	116	22	83	150
ChainTaskset	SWL	1, 8	34	76	124	34	87	136
	SWH	1, 9	61	114	163	61	125	175
	NSL	8	34	76	124	34	87	136
	NSH	9	55	108	157	55	119	169
GetTasksetRef			10	10	10	10	10	10
MergeTaskset			30	30	30	30	30	30
AssignTaskset			10	10	10	10	10	10
RemoveTaskset			32	32	32	32	32	32
TestSubTaskset			43	43	43	43	43	43
TestEquivalentTaskset			36	36	36	36	36	36
TickSchedule	SW	1	102	97	97	97	97	97
	NS		87	82	82	82	82	82
	KL	2	73	69	69	69	69	69
AdvanceSchedule	SW	1	95	90	90	90	90	90
	NS		80	75	75	75	75	75
	KL	2	70	65	65	65	65	65
StartSchedule			50	50	50	50	50	50
StopSchedule			34	34	34	34	34	34
GetScheduleStatus			60	60	60	60	60	60
GetScheduleValue			44	44	44	44	44	44
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			10	10	10	10	10	10
SetArrivalpointDelay			6	6	6	6	6	6
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			6	6	6	6	6	6

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestArrivalpointWritable			22	22	22	22	22	22
GetExecutionTime			3	3	3	3	3	3
GetLargestExecutionTime			8	8	8	8	8	8
ResetLargestExecutionTime			2	2	2	2	2	2
GetStackOffset			18	18	18	18	18	18
bit_mangling			47	47	47	47	47	47
ipl_mangling			29	29	29	29	29	29
Context save/restore			30	30	30	30	30	30
Interrupt support			12	12	12	12	12	12
Floating point support			18	18	18	18	18	18

Timing

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	76	125	161	82	131	185
	NS		64	113	149	70	119	173
	KL	2	42	104	140	48	110	162
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	67	116	153	73	122	177
	SWH	1, 9	90	137	173	96	143	198
	NSL	8	67	116	153	73	122	177
	NSH	9	84	131	167	90	137	192
Schedule			60	60	77	60	60	77
GetTaskID			22	22	22	22	22	22
GetTaskState			55	55	55	71	71	71
EnableAllInterrupts			6	6	6	6	6	6
DisableAllInterrupts			13	13	13	13	13	13
ResumeAllInterrupts			15	15	15	15	15	15
SuspendAllInterrupts			23	23	23	23	23	23
ResumeOSInterrupts			15	15	15	15	15	15
SuspendOSInterrupts			29	29	29	29	29	29

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetResource	Task	7	21	21	27	21	21	27
	Combined	6	54	54	54	54	54	54
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	50	50	50	50	50	50
	Combined	6	91	91	91	91	91	91
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	85	85	159
	NS		n/a	n/a	n/a	73	73	147
	NS1i	10	n/a	n/a	n/a	31	n/a	n/a
	KL	2	n/a	n/a	n/a	63	63	137
	KL1i	2, 10	n/a	n/a	n/a	13	n/a	n/a
ClearEvent			n/a	n/a	n/a	19	19	19
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	189	189	341
	fp	11	n/a	n/a	n/a	211	211	389
	1i	10	n/a	n/a	n/a	46	n/a	n/a
GetAlarmBase			32	32	32	32	32	32
GetAlarm			67	67	67	67	67	67
SetRelAlarm			87	87	87	87	87	87
SetAbsAlarm			104	104	104	104	104	104
CancelAlarm			49	49	49	49	49	49
InitCounter			37	37	37	37	37	37
GetCounterValue			43	43	43	43	43	43
osek_tick_alarm	<default>		49	49	49	49	49	49
	KL	2	29	29	29	29	29	29
osek_incr_counter			29	29	29	29	29	29
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			115	115	115	115	115	115
ShutdownOS	NoHook	12	11	11	11	11	11	11
	Hook	13	17	17	17	17	17	17
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			12	12	12	12	12	12
StopCOM			9	9	9	9	9	9
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	40	40	40	123	123	123

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	123	123	123	123	123	123
GetMessageResource			20	20	20	20	20	20
ReleaseMessageResource			18	18	18	18	18	18
GetMessageStatus			37	37	37	37	37	37
SendMessage	SW CCCA	1, 14	59	59	59	162	162	162
	SW CCCB	1, 15	144	144	144	162	162	162
	NS CCCA	14	59	59	59	162	162	162
	NS CCCB	15	144	144	144	162	162	162
	KL CCCA	2, 14	38	38	38	164	164	164
	KL CCCB	2, 15	151	151	151	164	164	164
main_dispatch	NoHook	12	125	125	163	125	125	163
	Hook	13	154	154	194	154	154	194
sub_dispatch	B1LF	19	9	9	9	9	9	9
	B1HI	20	60	60	60	60	60	60
	B1HF	21	68	68	68	68	68	68
	B2LI	22	n/a	32	61	n/a	32	61
	B2LF	23	n/a	37	67	n/a	37	67
	B2HI	24	n/a	87	150	n/a	87	150
	B2HF	25	n/a	93	156	n/a	93	156
	E1HI	26	n/a	n/a	n/a	247	247	314
	E1HF	27	n/a	n/a	n/a	253	253	320
	E2HI	28	n/a	n/a	n/a	n/a	n/a	314
	E2HF	29	n/a	n/a	n/a	n/a	n/a	320
ErrorHook support		16	17	17	17	17	17	17
	ServiceID	17	22	22	22	22	22	22
	Parameters	18	37	37	37	37	37	37
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	32	32	32	32	32	32
Timing_termination		4	49	49	49	49	49	49
ActivateTaskset	SW	1	42	85	134	50	104	154
	NS		30	73	122	38	92	142
	KL	2	14	65	116	22	83	150
ChainTaskset	SWL	1, 8	34	76	124	34	87	136
	SWH	1, 9	61	114	163	61	125	175
	NSL	8	34	76	124	34	87	136
	NSH	9	55	108	157	55	119	169
GetTasksetRef			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	
MergeTaskset			30	30	30	30	30	30
AssignTaskset			10	10	10	10	10	10
RemoveTaskset			32	32	32	32	32	32
TestSubTaskset			43	43	43	43	43	43
TestEquivalentTaskset			36	36	36	36	36	36
TickSchedule	SW	1	102	97	97	97	97	97
	NS		87	82	82	82	82	82
	KL	2	73	69	69	69	69	69
AdvanceSchedule	SW	1	95	90	90	90	90	90
	NS		80	75	75	75	75	75
	KL	2	70	65	65	65	65	65
StartSchedule			50	50	50	50	50	50
StopSchedule			34	34	34	34	34	34
GetScheduleStatus			60	60	60	60	60	60
GetScheduleValue			44	44	44	44	44	44
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			10	10	10	10	10	10
SetArrivalpointDelay			6	6	6	6	6	6
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			6	6	6	6	6	6
TestArrivalpointWritable			22	22	22	22	22	22
GetExecutionTime			49	49	49	49	49	49
GetLargestExecutionTime			14	14	14	14	14	14
ResetLargestExecutionTime			11	11	11	11	11	11
GetStackOffset			18	18	18	18	18	18
bit_mangling			47	47	47	47	47	47
ipl_mangling			29	29	29	29	29	29
Context save/restore			30	30	30	30	30	30
Interrupt support			87	87	87	87	87	87
Floating point support			18	18	18	18	18	18

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	138	187	224	144	193	248
	NS		179	230	266	185	236	290
	KL	2	117	180	217	123	186	241
TerminateTask	LExt	3	73	73	73	73	73	73
	H	5	94	94	94	94	94	94
ChainTask	SWL	1, 8	163	212	248	169	218	272
	SWH	1, 9	192	239	276	198	245	300
	NSL	8	216	271	307	225	277	331
	NSH	9	237	289	326	246	296	351
Schedule			120	120	137	120	120	137
GetTaskID			32	32	32	32	32	32
GetTaskState			141	141	141	147	147	147
EnableAllInterrupts			17	17	17	17	17	17
DisableAllInterrupts			23	23	23	23	23	23
ResumeAllInterrupts			47	47	47	47	47	47
SuspendAllInterrupts			33	33	33	33	33	33
ResumeOSInterrupts			47	47	47	47	47	47
SuspendOSInterrupts			39	39	39	39	39	39
GetResource	Task	7	235	235	204	235	235	204
	Combined	6	208	208	208	208	208	208
	CLEx	3	201	201	201	201	201	201
ReleaseResource	Task	7	195	195	195	195	195	195
	Combined	6	229	229	229	229	229	229
	CLEx	3	184	184	184	184	184	184
SetEvent	SW	1	n/a	n/a	n/a	179	179	254
	NS		n/a	n/a	n/a	219	219	291
	NS1i	10	n/a	n/a	n/a	123	n/a	n/a
	KL	2	n/a	n/a	n/a	173	173	235
	KL1i	2, 10	n/a	n/a	n/a	133	n/a	n/a
ClearEvent			n/a	n/a	n/a	85	85	85
GetEvent			n/a	n/a	n/a	120	120	120
WaitEvent	<default>		n/a	n/a	n/a	273	273	415
	fp	11	n/a	n/a	n/a	295	295	463

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	130	n/a	n/a
GetAlarmBase			89	89	89	89	89	89
GetAlarm			98	98	98	98	98	98
SetRelAlarm			146	146	146	146	146	146
SetAbsAlarm			161	161	161	161	161	161
CancelAlarm			81	81	81	81	81	81
InitCounter			136	136	136	136	136	136
GetCounterValue			99	99	99	99	99	99
osek_tick_alarm	<default>		49	49	49	49	49	49
	KL	2	29	29	29	29	29	29
osek_incr_counter			29	29	29	29	29	29
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			125	125	125	125	125	125
ShutdownOS	NoHook	12	16	16	16	16	16	16
	Hook	13	22	22	22	22	22	22
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			22	22	22	22	22	22
StopCOM			24	24	24	24	24	24
ReadFlag			15	15	15	15	15	15
ResetFlag			16	16	16	16	16	16
ReceiveMessage	CCCA	14	79	79	79	162	162	162
	CCCB	15	162	162	162	162	162	162
GetMessageResource			58	58	58	58	58	58
ReleaseMessageResource			58	58	58	58	58	58
GetMessageStatus			67	67	67	67	67	67
SendMessage	SW CCCA	1, 14	105	105	105	209	209	209
	SW CCCB	1, 15	191	191	191	209	209	209
	NS CCCA	14	105	105	105	209	209	209
	NS CCCB	15	191	191	191	209	209	209
	KL CCCA	2, 14	97	97	97	219	219	219
	KL CCCB	2, 15	204	204	204	219	219	219
main_dispatch	NoHook	12	125	125	163	125	125	163
	Hook	13	154	154	194	154	154	194
sub_dispatch	B1LF	19	9	9	9	9	9	9
	B1HI	20	60	60	60	60	60	60
	B1HF	21	68	68	68	68	68	68

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
	B2LI	22	n/a	32	61	n/a	32	61
	B2LF	23	n/a	37	67	n/a	37	67
	B2HI	24	n/a	87	150	n/a	87	150
	B2HF	25	n/a	93	156	n/a	93	156
	E1HI	26	n/a	n/a	n/a	247	247	314
	E1HF	27	n/a	n/a	n/a	253	253	320
	E2HI	28	n/a	n/a	n/a	n/a	n/a	314
	E2HF	29	n/a	n/a	n/a	n/a	n/a	320
ErrorHook support		16	39	39	39	39	39	39
	ServiceID	17	44	44	44	44	44	44
	Parameters	18	59	59	59	59	59	59
validity_checks		3	43	43	43	43	43	43
Timing_dispatch		4	32	32	32	32	32	32
Timing_termination		4	49	49	49	49	49	49
ActivateTaskset	SW	1	188	241	289	202	268	325
	NS		229	281	330	243	308	366
	KL	2	166	219	265	180	247	306
ChainTaskset	SWL	1, 8	237	289	337	246	309	367
	SWH	1, 9	273	336	384	282	356	415
	NSL	8	288	340	389	295	358	416
	NSH	9	318	381	430	325	399	458
GetTasksetRef			91	91	91	91	91	91
MergeTaskset			135	135	135	135	135	135
AssignTaskset			113	113	113	113	113	113
RemoveTaskset			137	137	137	137	137	137
TestSubTaskset			146	146	146	146	146	146
TestEquivalentTaskset			139	139	139	139	139	139
TickSchedule	SW	1	212	153	153	153	153	153
	NS		260	223	223	223	223	223
	KL	2	220	145	145	145	145	145
AdvanceSchedule	SW	1	219	160	160	160	160	160
	NS		263	230	230	230	230	230
	KL	2	226	158	158	158	158	158
StartSchedule			129	129	129	129	129	129
StopSchedule			83	83	83	83	83	83
GetScheduleStatus			110	110	110	110	110	110
GetScheduleValue			89	89	89	89	89	89

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleNext			57	57	57	57	57	57
SetScheduleNext			111	111	111	111	111	111
GetArrivalpointDelay			82	82	82	82	82	82
SetArrivalpointDelay			93	93	93	93	93	93
GetArrivalpointTasksetRef			78	78	78	78	78	78
GetArrivalpointNext			82	82	82	82	82	82
SetArrivalpointNext			119	119	119	119	119	119
TestArrivalpointWritable			94	94	94	94	94	94
GetExecutionTime			59	59	59	59	59	59
GetLargestExecutionTime			76	76	76	76	76	76
ResetLargestExecutionTime			66	66	66	66	66	66
GetStackOffset			18	18	18	18	18	18
bit_mangling			47	47	47	47	47	47
ipl_mangling			29	29	29	29	29	29
Context save/restore			30	30	30	30	30	30
Interrupt support			87	87	87	87	87	87
Floating point support			18	18	18	18	18	18

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.

Number	Note
15	Linked only if there are any flags, message queues, or message resources in the system, or COM status is requested.
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

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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
Service	Variant	No	Yes		No	Yes	
ActivateTask	SW	88	134	174	93	135	209
	NS	78	124	164	83	125	199
	KL	43	97	137	48	100	172
TerminateTask	LExt	0	0	0	0	0	0
	H	120	120	120	118	118	120
ChainTask	SWL	197	243	319	244	293	412
	SWH	253	295	372	299	344	464
	NSL	197	243	319	244	293	412
	NSH	248	290	367	294	339	459
Schedule	SW	79	79	92	79	79	92
GetTaskID		32	32	32	31	31	32
GetTaskState		90	90	89	102	103	103
EnableAllInterrupts		17	17	17	17	17	17
DisableAllInterrupts		28	28	28	28	28	28
ResumeAllInterrupts		31	31	31	31	31	31
SuspendAllInterrupts		38	38	38	38	38	38
ResumeOSInterrupts		31	31	31	31	31	31
SuspendOSInterrupts		38	38	38	38	38	38
GetResource	Task	42	39	41	39	39	44
	Combined	49	46	46	46	46	49
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	70	68	68	68	68	70
	Combined	63	60	60	60	60	63
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	107	107	112
	NS	n/a	n/a	n/a	107	107	112
	KL	n/a	n/a	n/a	83	83	88
ClearEvent		n/a	n/a	n/a	32	32	34
GetEvent		n/a	n/a	n/a	34	34	36
WaitEvent	<default>	n/a	n/a	n/a	295	306	383
	fp	n/a	n/a	n/a	304	316	393
GetAlarmBase		127	127	127	127	127	126
GetAlarm		100	100	100	100	100	97

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		106	106	106	106	106	103
SetAbsAlarm		105	105	105	105	105	102
CancelAlarm		66	66	66	66	66	65
InitCounter		74	74	74	74	74	72
GetCounterValue		75	75	75	75	75	73
osek_tick_alarm	<default>	101	101	101	101	101	100
	KL	51	51	51	51	51	50
osek_incr_counter		17	17	17	17	17	16
GetActiveApplicationMode		7	7	7	7	7	7
StartOS		292	292	292	306	306	292
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	28	28	28	28	28	28
InitCOM		14	14	14	14	14	14
CloseCOM		14	14	14	14	14	14
StartCOM		40	40	40	235	235	235
StopCOM		22	22	22	22	22	22
ReadFlag		n/a	n/a	n/a	15	15	15
ResetFlag		n/a	n/a	n/a	11	11	11
ReceiveMessage		68	68	68	200	200	209
GetMessageResource		n/a	n/a	n/a	88	88	90
ReleaseMessageResource		n/a	n/a	n/a	104	104	107
GetMessageStatus		n/a	n/a	n/a	47	47	48
SendMessage	SW	171	217	257	310	352	434
	NS	158	204	244	297	339	421
	KL	89	143	183	254	306	386
ActivateTaskset	SW	71	454	521	79	460	546
	NS	58	441	508	66	447	533
	KL	26	573	641	34	579	684
	SW2	71	454	521	79	460	546
	NS2	58	441	508	66	447	533
	KL2	26	573	641	34	579	684
ChainTaskset	SWL	192	572	673	233	618	748
	SWH	247	628	731	286	672	806
	NSL	192	572	673	233	618	748
	NSH	242	623	726	281	667	801
GetTasksetRef		31	31	30	30	31	31

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		67	67	67	67	67	67
AssignTaskset		29	29	29	29	29	29
RemoveTaskset		69	69	69	69	69	69
TestSubTaskset		81	81	81	81	81	81
TestEquivalentTaskset		72	72	72	72	72	72
TickSchedule	SW	149	709	784	167	725	824
	NS	133	693	768	151	709	808
	KL	99	665	740	123	681	780
	SW2	149	710	778	167	712	821
	NS2	133	694	762	151	696	805
	KL2	99	666	734	123	668	777
AdvanceSchedule	SW	127	684	759	151	709	799
	NS	111	668	743	135	693	783
	KL	87	644	719	111	669	759
	SW2	127	685	753	151	696	796
	NS2	111	669	737	135	680	780
	KL2	87	645	713	111	656	756
StartSchedule		85	85	85	87	87	85
StopSchedule		68	68	68	70	70	68
GetScheduleStatus		89	89	89	87	87	89
GetScheduleValue		79	79	79	80	80	79
GetScheduleNext		32	32	32	33	33	32
SetScheduleNext		29	29	29	30	30	29
GetArrivalpointDelay		29	29	29	30	30	29
SetArrivalpointDelay		26	26	26	26	26	26
GetArrivalpointTasksetRef		26	26	26	26	26	26
GetArrivalpointNext		29	29	29	29	29	29
SetArrivalpointNext		26	26	26	26	26	26
TestArrivalpointWritable		35	35	35	35	35	35
GetExecutionTime		12	12	12	12	12	12
GetLargestExecutionTime		28	28	28	27	27	28
ResetLargestExecutionTime		15	15	15	14	14	15
GetStackOffset		30	30	30	30	30	30

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	88	127	183	94	134	197
	NS	78	117	173	84	124	187
	KL	43	90	146	49	99	160
TerminateTask	LExt	0	0	0	0	0	0
	H	269	262	262	269	269	262
ChainTask	SWL	372	396	513	430	462	567
	SWH	429	449	565	487	516	620
	NSL	372	396	513	430	462	567
	NSH	424	444	560	482	511	615
Schedule	SW	81	77	89	81	81	89
GetTaskID		32	31	31	32	32	31
GetTaskState		90	89	90	103	102	102
EnableAllInterrupts		17	17	17	17	17	17
DisableAllInterrupts		28	28	28	28	28	28
ResumeAllInterrupts		31	31	31	31	31	31
SuspendAllInterrupts		38	38	38	38	38	38
ResumeOSInterrupts		31	31	31	31	31	31
SuspendOSInterrupts		38	38	38	38	38	38
GetResource	Task	42	42	43	42	42	40
	Combined	49	49	49	49	49	46
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	70	70	70	70	70	68
	Combined	63	63	63	63	63	60
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	112	112	107
	NS	n/a	n/a	n/a	112	112	107
	KL	n/a	n/a	n/a	88	88	83
ClearEvent		n/a	n/a	n/a	34	34	32
GetEvent		n/a	n/a	n/a	36	36	34
WaitEvent	<default>	n/a	n/a	n/a	456	442	508
	fp	n/a	n/a	n/a	466	451	517
GetAlarmBase		127	126	126	126	126	127
GetAlarm		100	97	97	97	97	100

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		106	103	103	103	103	106
SetAbsAlarm		105	102	102	102	102	105
CancelAlarm		66	65	65	65	65	66
InitCounter		74	72	72	72	72	74
GetCounterValue		75	73	73	73	73	75
osek_tick_alarm	<default>	101	100	100	100	100	101
	KL	51	50	50	50	50	51
osek_incr_counter		17	16	16	16	16	17
GetActiveApplicationMode		7	7	7	7	7	7
StartOS		565	602	602	565	565	601
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	28	28	28	28	28	28
InitCOM		14	14	14	14	14	14
CloseCOM		14	14	14	14	14	14
StartCOM		40	40	40	235	235	235
StopCOM		22	22	22	22	22	22
ReadFlag		n/a	n/a	n/a	15	15	15
ResetFlag		n/a	n/a	n/a	11	11	11
ReceiveMessage		68	66	66	209	209	200
GetMessageResource		n/a	n/a	n/a	91	91	87
ReleaseMessageResource		n/a	n/a	n/a	107	107	104
GetMessageStatus		n/a	n/a	n/a	48	48	47
SendMessage	SW	171	206	262	319	359	414
	NS	158	193	249	306	346	401
	KL	89	132	188	263	313	366
ActivateTaskset	SW	71	451	523	79	462	533
	NS	58	438	510	66	449	520
	KL	26	570	643	34	581	671
	SW2	71	451	523	79	462	533
	NS2	58	438	510	66	449	520
	KL2	26	570	643	34	581	671
ChainTaskset	SWL	367	728	859	419	791	901
	SWH	423	783	916	475	848	958
	NSL	367	728	859	419	791	901
	NSH	418	778	911	470	843	953
GetTasksetRef		31	30	31	31	30	30

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		67	67	67	67	67	67
AssignTaskset		29	29	29	29	29	29
RemoveTaskset		69	69	69	69	69	69
TestSubTaskset		81	81	81	81	81	81
TestEquivalentTaskset		72	72	72	72	72	72
TickSchedule	SW	149	704	770	171	731	820
	NS	133	688	754	155	715	804
	KL	99	660	726	127	687	776
	SW2	149	703	776	171	718	804
	NS2	133	687	760	155	702	788
	KL2	99	659	732	127	674	760
AdvanceSchedule	SW	127	688	754	146	706	804
	NS	111	672	738	130	690	788
	KL	87	648	714	106	666	764
	SW2	127	687	760	146	693	788
	NS2	111	671	744	130	677	772
	KL2	87	647	720	106	653	748
StartSchedule		85	87	87	85	85	87
StopSchedule		68	70	70	68	68	70
GetScheduleStatus		89	87	87	89	89	87
GetScheduleValue		79	80	80	79	79	80
GetScheduleNext		32	33	33	32	32	33
SetScheduleNext		29	30	30	29	29	30
GetArrivalpointDelay		29	30	30	29	29	30
SetArrivalpointDelay		26	26	26	26	26	26
GetArrivalpointTasksetRef		26	26	26	26	26	26
GetArrivalpointNext		29	29	29	29	29	29
SetArrivalpointNext		26	26	26	26	26	26
TestArrivalpointWritable		35	35	35	35	35	35
GetExecutionTime		88	87	87	88	88	86
GetLargestExecutionTime		40	37	37	40	40	37
ResetLargestExecutionTime		27	24	24	27	27	24
GetStackOffset		30	30	30	30	30	30

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	438	478	540	462	486	549
	NS	473	513	574	497	522	584
	KL	411	453	515	435	463	524
TerminateTask	LExt	282	282	286	286	282	282
	H	327	327	332	332	327	327
ChainTask	SWL	758	798	914	841	863	971
	SWH	814	851	968	898	915	1024
	NSL	801	843	958	886	909	1016
	NSH	852	891	1007	938	956	1064
Schedule	SW	103	103	116	104	103	116
GetTaskID		39	39	40	40	39	39
GetTaskState		454	454	474	481	461	460
EnableAllInterrupts		31	31	31	31	31	31
DisableAllInterrupts		36	36	36	36	36	36
ResumeAllInterrupts		45	45	45	45	45	45
SuspendAllInterrupts		46	46	46	46	46	46
ResumeOSInterrupts		45	45	45	45	45	45
SuspendOSInterrupts		46	46	46	46	46	46
GetResource	Task	651	683	356	805	747	453
	Combined	313	331	312	429	406	406
	CLEx	375	391	375	490	468	468
ReleaseResource	Task	334	352	332	449	427	427
	Combined	304	321	304	420	397	397
	CLEx	338	353	338	452	431	431
SetEvent	SW	n/a	n/a	n/a	487	462	462
	NS	n/a	n/a	n/a	509	484	484
	KL	n/a	n/a	n/a	489	464	453
ClearEvent		n/a	n/a	n/a	70	68	68
GetEvent		n/a	n/a	n/a	443	420	420
WaitEvent	<default>	n/a	n/a	n/a	508	508	557
	fp	n/a	n/a	n/a	518	518	566
GetAlarmBase		371	371	385	371	385	385
GetAlarm		342	342	358	342	358	358

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		374	374	394	374	394	394
SetAbsAlarm		367	367	386	367	386	386
CancelAlarm		310	310	324	310	324	324
InitCounter		488	488	522	488	522	522
GetCounterValue		287	287	300	287	300	300
osek_tick_alarm	<default>	100	100	101	100	101	101
	KL	50	50	51	50	51	51
osek_incr_counter		16	16	17	16	17	17
GetActiveApplicationMode		7	7	7	7	7	7
StartOS		587	588	628	629	587	587
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	32	32	32	32	32	32
InitCOM		14	14	14	14	14	14
CloseCOM		14	14	14	14	14	14
StartCOM		52	52	52	247	247	247
StopCOM		33	33	33	33	33	33
ReadFlag		n/a	n/a	n/a	32	32	32
ResetFlag		n/a	n/a	n/a	29	29	29
ReceiveMessage		224	224	233	367	365	365
GetMessageResource		n/a	n/a	n/a	666	651	651
ReleaseMessageResource		n/a	n/a	n/a	640	625	625
GetMessageStatus		n/a	n/a	n/a	205	211	211
SendMessage	SW	675	715	788	845	868	931
	NS	707	747	819	877	901	963
	KL	637	679	752	825	852	913
ActivateTaskset	SW	547	949	1015	560	958	1037
	NS	575	977	1043	588	986	1065
	KL	435	842	907	448	846	933
	SW2	547	949	1015	560	958	1037
	NS2	575	977	1043	588	986	1065
	KL2	435	842	907	448	846	933
ChainTaskset	SWL	903	1307	1428	976	1368	1492
	SWH	960	1363	1484	1033	1424	1548
	NSL	944	1348	1469	1015	1407	1531
	NSH	996	1399	1520	1067	1458	1582
GetTasksetRef		389	389	409	410	390	389

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		113	113	113	113	113	113
AssignTaskset		81	81	81	81	81	81
RemoveTaskset		113	113	113	113	113	113
TestSubTaskset		127	127	127	127	127	127
TestEquivalentTaskset		118	118	118	118	118	118
TickSchedule	SW	209	1001	1068	602	1034	1119
	NS	250	1040	1107	641	1073	1158
	KL	200	987	1054	588	1020	1105
	SW2	209	999	1061	602	1003	1090
	NS2	250	1038	1100	641	1042	1129
	KL2	200	985	1047	588	989	1076
AdvanceSchedule	SW	190	982	1058	592	1015	1100
	NS	226	1018	1096	630	1051	1136
	KL	180	962	1039	573	995	1080
	SW2	190	980	1051	592	984	1071
	NS2	226	1016	1089	630	1020	1107
	KL2	180	960	1032	573	964	1051
StartSchedule		120	120	122	122	120	120
StopSchedule		84	84	86	86	84	84
GetScheduleStatus		105	105	103	103	105	105
GetScheduleValue		95	95	96	96	95	95
GetScheduleNext		54	54	55	55	54	54
SetScheduleNext		83	83	86	86	83	83
GetArrivalpointDelay		59	59	60	60	59	59
SetArrivalpointDelay		64	64	63	63	64	64
GetArrivalpointTasksetRef		49	49	49	49	49	49
GetArrivalpointNext		53	53	53	53	53	53
SetArrivalpointNext		83	83	84	84	83	83
TestArrivalpointWritable		59	59	59	59	59	59
GetExecutionTime		95	96	94	95	95	95
GetLargestExecutionTime		386	386	406	406	386	386
ResetLargestExecutionTime		366	366	386	386	366	366
GetStackOffset		30	30	30	30	30	30

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	15	15	15	15	15	15
	Cat 2	18	18	18	18	18	18

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	15	15	15	15	15	15
	Cat 2	107	102	102	107	107	101

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	15	15	15	15	15	15
	Cat 2	106	107	101	102	106	106

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.

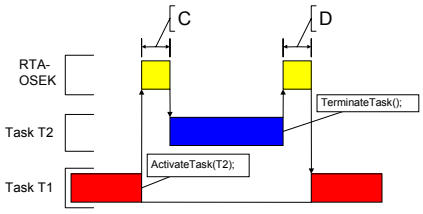

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

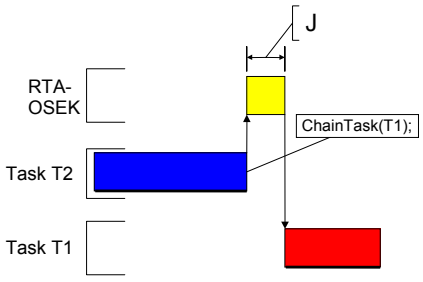

Figure 2: Task Chaining

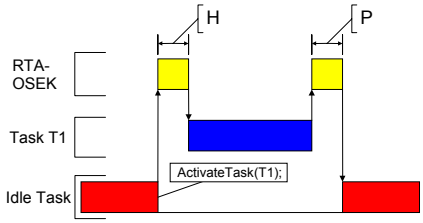

Figure 3: Task Activation from Idle Task

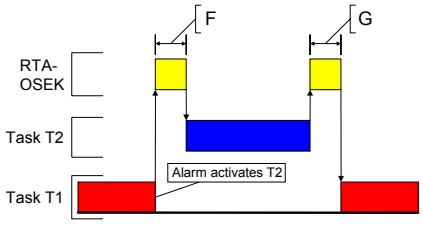

Figure 4: Task Activation from an Alarm

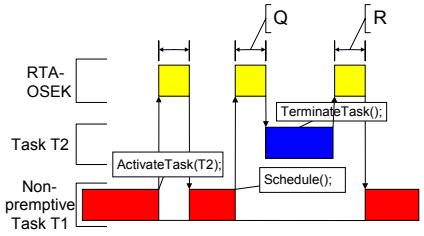

Figure 5: Non-Premptive Task Calls Schedule()

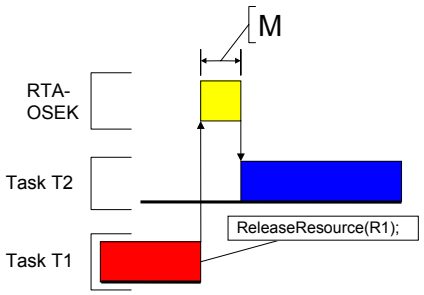

Figure 6: Blocked Task Activated by ReleaseResource()

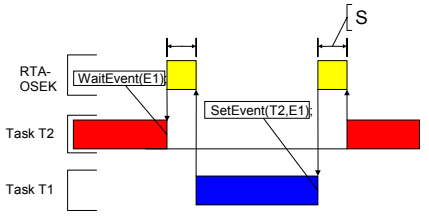

Figure 7: Waiting Task Activated by SetEvent()

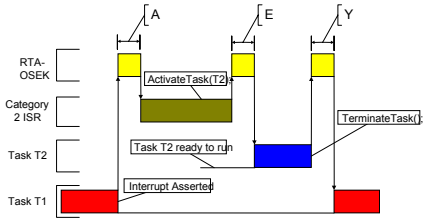

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	70	90	128	69	92	123
Figure 1: D	Heavy, Basic/Extended	120	137	165	147	151	185
ChainTask	Light, Basic	150	199	288	151	195	323
Figure 2: J	Heavy, Basic/Extended	341	404	521	368	413	575
Pre-emption	Light, Basic	142	191	297	143	195	304
Figure 1: C	Heavy, Basic/Extended	192	238	317	240	289	410
From idle task	Light, Basic	142	191	297	143	195	304
Figure 3: H	Heavy, Basic/Extended	192	238	317	240	289	410
Triggered by alarm	Light, Basic	239	288	394	239	291	401
Figure 4: F	Heavy, Basic/Extended	289	335	414	336	385	507
Schedule	Light, Basic	124	132	196	118	132	190
Figure 5: Q	Heavy, Basic/Extended	174	174	226	215	223	283
Release resource	Light, Basic	136	141	192	129	142	190
Figure 6: M	Heavy, Basic/Extended	186	183	222	226	233	283
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	371	384	551

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
From category 2 ISR	Light, Basic	96	104	155	92	105	150
Figure 8: E	Heavy, Basic/Extended	146	146	185	189	196	243

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	218	237	265	218	234	273
Figure 1: D	Heavy, Basic/Extended	269	277	317	299	292	321
ChainTask	Light, Basic	332	369	476	338	382	481
Figure 2: J	Heavy, Basic/Extended	673	713	860	709	742	870
Pre-emption	Light, Basic	250	292	384	256	291	417
Figure 1: C	Heavy, Basic/Extended	294	322	439	352	384	491
From idle task	Light, Basic	250	292	384	256	291	417
Figure 3: H	Heavy, Basic/Extended	294	322	439	352	384	491
Triggered by alarm	Light, Basic	347	388	480	353	388	513
Figure 4: F	Heavy, Basic/Extended	391	418	535	449	481	587
Schedule	Light, Basic	232	226	289	232	229	298
Figure 5: Q	Heavy, Basic/Extended	276	261	334	328	317	377
Release resource	Light, Basic	244	242	293	244	244	294
Figure 6: M	Heavy, Basic/Extended	288	277	338	340	332	373
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	474	461	613
From category 2 ISR	Light, Basic	357	349	400	357	357	404
Figure 8: E	Heavy, Basic/Extended	401	384	445	453	445	483

Extended

Configuration Events Shared Task Priorities Multiple Task Activations	Task Attributes	Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Normal termination	Light, Basic	282	308	343	286	308	344
Figure 1: D	Heavy, Basic/Extended	327	342	372	362	363	386
ChainTask	Light, Basic	718	771	883	749	762	885
Figure 2: J	Heavy, Basic/Extended	1116	1180	1324	1183	1193	1339
Pre-emption	Light, Basic	589	644	762	615	642	771
Figure 1: C	Heavy, Basic/Extended	632	674	782	711	737	845
From idle task	Light, Basic	589	644	762	615	642	771
Figure 3: H	Heavy, Basic/Extended	632	674	782	711	737	845
Triggered by alarm	Light, Basic	685	740	859	712	738	867
Figure 4: F	Heavy, Basic/Extended	728	770	879	808	833	941
Schedule	Light, Basic	244	252	320	247	256	327
Figure 5: Q	Heavy, Basic/Extended	287	287	350	343	348	406
Release resource	Light, Basic	490	518	549	607	591	649
Figure 6: M	Heavy, Basic/Extended	533	553	579	703	683	728
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	843	823	974
From category 2 ISR	Light, Basic	355	365	416	357	363	421
Figure 8: E	Heavy, Basic/Extended	398	400	446	453	455	500

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	17	17	17	17	17	17
BCC1 lightweight, floating-point	19	19	19	19	19	19
BCC1 heavyweight, integer	24	24	24	24	24	24
BCC1 heavyweight, floating-point	24	24	24	24	24	24
BCC2 lightweight, integer	n/a	19	21	n/a	19	21
BCC2 lightweight, floating-point	n/a	19	21	n/a	19	21
BCC2 heavyweight, integer	n/a	24	28	n/a	24	28
BCC2 heavyweight, floating-point	n/a	24	28	n/a	24	28
ECC1 heavyweight, integer	n/a	n/a	n/a	34	34	34
ECC1 heavyweight, floating-point	n/a	n/a	n/a	34	34	34
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	38
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	38
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	17	17	17	17	17	17
BCC1 lightweight, floating-point	19	19	19	19	19	19
BCC1 heavyweight, integer	24	24	24	24	24	24
BCC1 heavyweight, floating-point	24	24	24	24	24	24
BCC2 lightweight, integer	n/a	19	21	n/a	19	21
BCC2 lightweight, floating-point	n/a	19	21	n/a	19	21
BCC2 heavyweight, integer	n/a	24	28	n/a	24	28
BCC2 heavyweight, floating-point	n/a	24	28	n/a	24	28
ECC1 heavyweight, integer	n/a	n/a	n/a	34	34	34
ECC1 heavyweight, floating-point	n/a	n/a	n/a	34	34	34
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	38
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	38

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	21	21	21	21	21	21
BCC1 lightweight, floating-point	23	23	23	23	23	23
BCC1 heavyweight, integer	28	28	28	28	28	28
BCC1 heavyweight, floating-point	28	28	28	28	28	28
BCC2 lightweight, integer	n/a	23	25	n/a	23	25
BCC2 lightweight, floating-point	n/a	23	25	n/a	23	25
BCC2 heavyweight, integer	n/a	28	32	n/a	28	32
BCC2 heavyweight, floating-point	n/a	28	32	n/a	28	32
ECC1 heavyweight, integer	n/a	n/a	n/a	38	38	38
ECC1 heavyweight, floating-point	n/a	n/a	n/a	38	38	38
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	42
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	42
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	21	21	21	21	21	21
BCC1 lightweight, floating-point	23	23	23	23	23	23
BCC1 heavyweight, integer	28	28	28	28	28	28
BCC1 heavyweight, floating-point	28	28	28	28	28	28
BCC2 lightweight, integer	n/a	23	25	n/a	23	25
BCC2 lightweight, floating-point	n/a	23	25	n/a	23	25
BCC2 heavyweight, integer	n/a	28	32	n/a	28	32
BCC2 heavyweight, floating-point	n/a	28	32	n/a	28	32
ECC1 heavyweight, integer	n/a	n/a	n/a	38	38	38
ECC1 heavyweight, floating-point	n/a	n/a	n/a	38	38	38
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	42
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	42

Extended

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		21	21	21	21	21	21
BCC1 lightweight, floating-point		23	23	23	23	23	23
BCC1 heavyweight, integer		28	28	28	28	28	28
BCC1 heavyweight, floating-point		28	28	28	28	28	28
BCC2 lightweight, integer		n/a	23	25	n/a	23	25
BCC2 lightweight, floating-point		n/a	23	25	n/a	23	25
BCC2 heavyweight, integer		n/a	28	32	n/a	28	32
BCC2 heavyweight, floating-point		n/a	28	32	n/a	28	32
ECC1 heavyweight, integer		n/a	n/a	n/a	38	38	38
ECC1 heavyweight, floating-point		n/a	n/a	n/a	38	38	38
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	42
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	42
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		21	21	21	21	21	21
BCC1 lightweight, floating-point		23	23	23	23	23	23
BCC1 heavyweight, integer		28	28	28	28	28	28
BCC1 heavyweight, floating-point		28	28	28	28	28	28
BCC2 lightweight, integer		n/a	23	25	n/a	23	25
BCC2 lightweight, floating-point		n/a	23	25	n/a	23	25
BCC2 heavyweight, integer		n/a	28	32	n/a	28	32
BCC2 heavyweight, floating-point		n/a	28	32	n/a	28	32
ECC1 heavyweight, integer		n/a	n/a	n/a	38	38	38
ECC1 heavyweight, floating-point		n/a	n/a	n/a	38	38	38
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	42
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	42

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.