RTA-OSEK

Binding Manual: HC12/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.
4F, 705 Bldg. 70-5
Yangjae-dong, Seocho-gu
Seoul 137-889, Korea

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: M00079-001

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 HC12/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 architecture offers three memory models: small, banked, and large. **This port is built for the banked memory model.**

2.1.1 Background

The HC(S)12 has a sixteen bit logical address space. Standard HC(S)12 instructions always use logical addresses. Within this logical address space there are up to three banked memory windows: one for code (which is always present), one for data, and an 'extra' window, e.g. for EEPROM. Whenever a standard HC(S)12 instruction references memory in the banked code window the contents of the `PPAGE` register are used to determine which of several code memory pages is currently visible in the memory window. The `DPAGE` and `EPAGE` registers, if they exist, are used for the data and extra banked memory windows respectively.

Placement of code and data into memory pages is controlled by the linker command file. See the "Segmentation" section of the "HC(S)12 Back End" chapter of the CodeWarrior Compiler Manual for details.

Note: when this manual refers to "unbanked ROM" it means Flash memory **not** in the `0x8000` to `0xBFFF` banked memory window.

2.1.2 Code Addresses

In the banked memory model, code is "far" by default. Far code addresses are three bytes wide, with the upper byte coming from the `PPAGE` register. The most significant byte indicates the page of flash memory that should be mapped into the banked memory window between `0x8000` and `0xBFFF`. The currently-mapped page is stored in the `PPAGE` register and the assembly instructions `CALL` and `RTC` maintain this implicitly. Therefore it is not necessary for an application to preserve the `PPAGE` register.

"Near" code addresses are only two bytes wide, and must be present in the sixteen bit logical address space at the time of calling. Near code must be placed in an unbanked memory area.

2.1.3 Data Addresses

In the banked memory model, data is "near" by default. Near data addresses are two bytes wide, and near data must be placed in an unbanked memory area. All data in the kernel is near and therefore it does not use the `DPAGE` and `EPAGE` registers.

If an application needs to modify the `DPAGE` and `EPAGE` registers then this can be achieved using the floating-point wrappers in `osfptgt.c`. Care must be taken to ensure that RTA-OSEK data is always visible in the 64KB logical address space.

2.1.4 Mixing Memory Models

Since the kernel was built and tested only with the banked memory model, applications interacting with it must ensure that application code matches the banked memory model's default pointer sizes, namely that code pointers are far and data pointers are near.

This applies to both application functions called by the kernel (e.g. task or ISR entry functions), and application functions which make RTA-OSEK API calls.

This should be achieved by compiling and assembling all application code using the banked memory model.

2.2 Compiler

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

Vendor	Freescale
Compiler	CodeWarrior HC12 C Compiler (chc12.exe)
Version	5.0.30 Build 6037

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

Option	Description
`-Cni`	Do not widen char parameters to integers.
`-Mb`	Build for the banked memory model.

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.
`-Mb`	Build for the banked memory model.

2.3 Assembler

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

Vendor	Freescale
Assembler	CodeWarrior HC12 Assembler (ahc12.exe)
Version	5.0.29 Build 6037

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

Option	Description
`-Mb`	Build for the banked memory model.

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.

The compulsory assembler options for `osgen.asm` are shown in the following table:

Option	Description
`-Mb`	Build for the banked memory model.

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

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

Sections	Rom/Ram	Description
os_pid	Unbanked ROM	RTA-OSEK read-only data
os_pird	Unbanked ROM	RTA-OSEK initialization data
os_intvec	Unbanked ROM	Vector table if generated by RTA-OSEK GUI
os_pir	Unbanked RAM	RTA-OSEK initialized data
os_pur	Unbanked RAM	RTA-OSEK uninitialized data
os_data_unbanked	Unbanked RAM	RTA-OSEK data
os_constdata_unbanked	Unbanked ROM	RTA-OSEK read-only data
os_text_unbanked	Unbanked ROM	RTA-OSEK interrupt handling code
os_text	ROM	RTA-OSEK code

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

C Library Functions	Description
_COPY	Structure copying routine
_FPCMP	Far pointer comparison
_LCMP	Long compare
_LCMP_P	Long compare with pointer

It is 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 both of 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 CodeWarrior 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 sixty-four 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.

`osStackOffsetType` is a scalar, representing the number of bytes on the stack, with C type: `unsigned long` .

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): 45

Timing

API max usage (bytes): 45

Extended

API max usage (bytes): 45

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

HC(S)12 instructions that push data onto the stack decrement the SP register before using it. Therefore the startup code provided with the compiler initializes the SP to the value of __SEG_END_SSTACK, which is a linker-defined constant equal to the address of the first byte beyond the stack area.

For example, if the linker command file specifies that the SSTACK section should be placed in memory at addresses 0x3000-0x3FFF, then the SP will be initialized to 0x4000. A subsequent stack 'push' will therefore write to memory at 0x3FFF (and not 0x4000).

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

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	107	107	112	191	191	196
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	172	172	177	256	256	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	51	51	51	51	51	51
	ROM	197	197	202	281	281	286
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	20	20	20	20	20	20
BCC1 Heavyweight task	RAM	2	2	2	2	2	2
	ROM	22	22	22	22	22	22
BCC2 task	RAM	n/a	3	5	n/a	3	5
	ROM	n/a	25	29	n/a	25	29
ECC1, Integer task	RAM	n/a	n/a	n/a	11	11	11
	ROM	n/a	n/a	n/a	32	32	32
ECC1, floating-point task	RAM	n/a	n/a	n/a	12	12	12
	ROM	n/a	n/a	n/a	32	32	32
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	36
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	36
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	28	28	28	28	28	28
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	41	41	41	41	41	41
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	9	9	9	9	9	9
	ROM	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	89	89	89	89	89	89
Message	RAM	11	11	11	31	31	31
	ROM	11	11	11	28	28	28
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
ScheduleTable	RAM	9	9	9	9	9	9
	ROM	100	100	100	100	100	100
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	9	9	9	9	9	9
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	11	11	11	11	11	11
	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

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	9	9	9	9	9	9
	ROM	27	27	27	27	27	27
BCC1 Heavyweight task	RAM	11	11	11	11	11	11
	ROM	29	29	29	29	29	29
BCC2 task	RAM	n/a	12	14	n/a	12	14
	ROM	n/a	32	36	n/a	32	36
ECC1, Integer task	RAM	n/a	n/a	n/a	20	20	20
	ROM	n/a	n/a	n/a	39	39	39
ECC1, floating-point task	RAM	n/a	n/a	n/a	21	21	21
	ROM	n/a	n/a	n/a	39	39	39

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	22
	ROM	n/a	n/a	n/a	n/a	n/a	43
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	23
	ROM	n/a	n/a	n/a	n/a	n/a	43
Category 2 ISR	RAM	9	9	9	9	9	9
	ROM	44	44	44	44	44	44
Category 2 ISR, floating-point	RAM	10	10	10	10	10	10
	ROM	52	52	52	52	52	52
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	9	9	9	9	9	9
	ROM	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	89	89	89	89	89	89
Message	RAM	11	11	11	31	31	31
	ROM	11	11	11	28	28	28
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
ScheduleTable	RAM	9	9	9	9	9	9
	ROM	100	100	100	100	100	100
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	9	9	9	9	9	9
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Schedule	RAM	11	11	11	11	11	11
	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

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	10	10	10	10	10	10
	ROM	31	31	31	31	31	31
BCC1 Heavyweight task	RAM	12	12	12	12	12	12
	ROM	31	31	31	31	31	31
BCC2 task	RAM	n/a	13	15	n/a	13	15
	ROM	n/a	34	38	n/a	34	38
ECC1, Integer task	RAM	n/a	n/a	n/a	21	21	21
	ROM	n/a	n/a	n/a	41	41	41
ECC1, floating-point task	RAM	n/a	n/a	n/a	22	22	22
	ROM	n/a	n/a	n/a	41	41	41
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	23
	ROM	n/a	n/a	n/a	n/a	n/a	45
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	24
	ROM	n/a	n/a	n/a	n/a	n/a	45
Category 2 ISR	RAM	10	10	10	10	10	10
	ROM	48	48	48	48	48	48
Category 2 ISR, floating-point	RAM	11	11	11	11	11	11
	ROM	56	56	56	56	56	56
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Linked resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Alarm	RAM	9	9	9	9	9	9
	ROM	32	32	32	32	32	32
Counter	RAM	4	4	4	4	4	4
	ROM	91	91	91	91	91	91
Message	RAM	11	11	11	31	31	31
	ROM	13	13	13	30	30	30
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
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
ScheduleTable	RAM	9	9	9	9	9	9
	ROM	100	100	100	100	100	100
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	9	9	9	9	9	9
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	15	15	15	15	15	15
	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

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.

Variant	Description
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	90	137	173	96	143	197
	NS		77	124	160	83	130	184
	KL	2	53	100	141	59	106	165
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	18	18	18	18	18	18
ChainTask	SWL	1, 8	80	127	164	86	133	188
	SWH	1, 9	104	149	186	110	155	210
	NSL	8	80	127	164	86	133	188
	NSH	9	98	143	180	104	149	204
Schedule			52	52	69	52	52	69
GetTaskID			22	22	22	22	22	22
GetTaskState			55	55	55	71	71	71
EnableAllInterrupts			7	7	7	7	7	7
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	42	42	42	42	42	42
	Combined	6	75	75	75	75	75	75
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	90	90	162
	NS		n/a	n/a	n/a	77	77	149
	NS1i	10	n/a	n/a	n/a	38	n/a	n/a

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	KL	2	n/a	n/a	n/a	67	67	139
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	26	26	26
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	168	168	325
	fp	11	n/a	n/a	n/a	192	192	376
	1i	10	n/a	n/a	n/a	17	n/a	n/a
GetAlarmBase			32	32	32	32	32	32
GetAlarm			81	81	81	81	81	81
SetRelAlarm			804	804	804	804	804	804
SetAbsAlarm			952	952	952	952	952	952
CancelAlarm			53	53	53	53	53	53
InitCounter			55	55	55	55	55	55
GetCounterValue			79	79	79	79	79	79
GetScheduleTableStatus		34	54	78	78	54	78	78
NextScheduleTable		34	76	201	201	76	201	201
StartScheduleTable		34	101	148	148	101	148	148
StopScheduleTable		34	69	87	87	69	87	87
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		5	5	5	5	5	5
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		27	27	27	27	27	27
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	25	25	25	25	25	25
Process container	Non-Yielding	33	13	13	13	13	13	13
osek_tick_alarm	<default>		76	76	76	76	76	76
	KL	2	59	59	59	59	59	59
osek_incr_counter			74	74	74	74	74	74
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			92	92	92	92	92	92
ShutdownOS	NoHook	12	11	11	11	11	11	11
	Hook	13	18	18	18	18	18	18
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			14	14	14	14	14	14
StopCOM			9	9	9	9	9	9

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
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	129	129	129
	CCCB	15	129	129	129	129	129	129
GetMessageResource			21	21	21	21	21	21
ReleaseMessageResource			19	19	19	19	19	19
GetMessageStatus			37	37	37	37	37	37
SendMessage	SW CCCA	1, 14	72	72	72	184	184	184
	SW CCCB	1, 15	163	163	163	184	184	184
	NS CCCA	14	72	72	72	184	184	184
	NS CCCB	15	163	163	163	184	184	184
	KL CCCA	2, 14	51	51	51	189	189	189
	KL CCCB	2, 15	166	166	166	189	189	189
main_dispatch	NoHook	12	78	78	109	78	78	109
	Hook	13	104	104	135	104	104	135
sub_dispatch	B1LF	19	17	17	17	17	17	17
	B1HI	20	54	54	54	54	54	54
	B1HF	21	62	62	62	62	62	62
	B2LI	22	n/a	39	68	n/a	39	68
	B2LF	23	n/a	47	76	n/a	47	76
	B2HI	24	n/a	140	202	n/a	140	202
	B2HF	25	n/a	148	210	n/a	148	210
	E1HI	26	n/a	n/a	n/a	233	233	301
	E1HF	27	n/a	n/a	n/a	241	241	309
	E2HI	28	n/a	n/a	n/a	n/a	n/a	301
	E2HF	29	n/a	n/a	n/a	n/a	n/a	309
ErrorHook support		16	18	18	18	18	18	18
	ServiceID	17	23	23	23	23	23	23
	Parameters	18	53	53	53	53	53	53
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	88	199	247	102	226	294
	NS		75	185	234	89	213	281
	KL	2	51	159	210	65	188	258
ChainTaskset	SWL	1, 8	85	205	253	97	226	291
	SWH	1, 9	119	250	299	131	271	337

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	NSL	8	85	205	253	97	226	291
	NSH	9	113	244	293	125	265	331
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	196	165	165	165	165	165
	NS		174	149	149	149	149	149
	KL	2	180	145	145	145	145	145
AdvanceSchedule	SW	1	156	129	129	129	129	129
	NS		140	113	113	113	113	113
	KL	2	135	101	101	101	101	101
StartSchedule			59	59	59	59	59	59
StopSchedule			36	36	36	36	36	36
GetScheduleStatus			72	72	72	72	72	72
GetScheduleValue			56	56	56	56	56	56
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			32	32	32	32	32	32
SetArrivalpointDelay			29	29	29	29	29	29
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			6	6	6	6	6	6
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			2	2	2	2	2	2
GetStackOffset			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	
Service name	Variant	Notes						
ActivateTask	SW	1	90	137	173	96	143	197
	NS		77	124	160	83	130	184
	KL	2	53	100	141	59	106	165
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	18	18	18	18	18	18
ChainTask	SWL	1, 8	80	127	164	86	133	188
	SWH	1, 9	104	149	186	110	155	210
	NSL	8	80	127	164	86	133	188
	NSH	9	98	143	180	104	149	204
Schedule			73	73	90	73	73	90
GetTaskID			22	22	22	22	22	22
GetTaskState			55	55	55	71	71	71
EnableAllInterrupts			7	7	7	7	7	7
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	63	63	63	63	63	63
	Combined	6	117	117	117	117	117	117
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	90	90	162
	NS		n/a	n/a	n/a	77	77	149
	NS1i	10	n/a	n/a	n/a	38	n/a	n/a
	KL	2	n/a	n/a	n/a	67	67	139
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	26	26	26
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	204	204	361
	fp	11	n/a	n/a	n/a	228	228	412

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	54	n/a	n/a
GetAlarmBase			32	32	32	32	32	32
GetAlarm			81	81	81	81	81	81
SetRelAlarm			804	804	804	804	804	804
SetAbsAlarm			952	952	952	952	952	952
CancelAlarm			53	53	53	53	53	53
InitCounter			55	55	55	55	55	55
GetCounterValue			79	79	79	79	79	79
GetScheduleTableStatus		34	54	78	78	54	78	78
NextScheduleTable		34	76	201	201	76	201	201
StartScheduleTable		34	101	148	148	101	148	148
StopScheduleTable		34	69	87	87	69	87	87
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		5	5	5	5	5	5
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		27	27	27	27	27	27
GetISRID		4	34	34	34	34	34	34
Process container	Yielding	32	25	25	25	25	25	25
Process container	Non-Yielding	33	13	13	13	13	13	13
osek_tick_alarm	<default>		76	76	76	76	76	76
	KL	2	59	59	59	59	59	59
osek_incr_counter			74	74	74	74	74	74
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			130	130	130	130	130	130
ShutdownOS	NoHook	12	11	11	11	11	11	11
	Hook	13	18	18	18	18	18	18
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			14	14	14	14	14	14
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	129	129	129
	CCCB	15	129	129	129	129	129	129
GetMessageResource			21	21	21	21	21	21
ReleaseMessageResource			19	19	19	19	19	19

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			37	37	37	37	37	37
SendMessage	SW CCCA	1, 14	72	72	72	184	184	184
	SW CCCB	1, 15	163	163	163	184	184	184
	NS CCCA	14	72	72	72	184	184	184
	NS CCCB	15	163	163	163	184	184	184
	KL CCCA	2, 14	51	51	51	189	189	189
	KL CCCB	2, 15	166	166	166	189	189	189
main_dispatch	NoHook	12	184	184	224	184	184	224
	Hook	13	215	215	255	215	215	255
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	64	64	64	64	64	64
	B1HF	21	74	74	74	74	74	74
	B2LI	22	n/a	33	62	n/a	33	62
	B2LF	23	n/a	41	70	n/a	41	70
	B2HI	24	n/a	146	209	n/a	146	209
	B2HF	25	n/a	154	217	n/a	154	217
	E1HI	26	n/a	n/a	n/a	262	262	329
	E1HF	27	n/a	n/a	n/a	270	270	337
	E2HI	28	n/a	n/a	n/a	n/a	n/a	329
	E2HF	29	n/a	n/a	n/a	n/a	n/a	337
ErrorHook support		16	18	18	18	18	18	18
	ServiceID	17	23	23	23	23	23	23
	Parameters	18	53	53	53	53	53	53
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	45	45	45	45	45	45
Timing_termination		4	85	85	85	85	85	85
ActivateTaskset	SW	1	88	199	247	102	226	294
	NS		75	185	234	89	213	281
	KL	2	51	159	210	65	188	258
ChainTaskset	SWL	1, 8	85	205	253	97	226	291
	SWH	1, 9	119	250	299	131	271	337
	NSL	8	85	205	253	97	226	291
	NSH	9	113	244	293	125	265	331
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

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestSubTaskset			43	43	43	43	43	43
TestEquivalentTaskset			36	36	36	36	36	36
TickSchedule	SW	1	196	165	165	165	165	165
	NS		174	149	149	149	149	149
	KL	2	180	145	145	145	145	145
AdvanceSchedule	SW	1	156	129	129	129	129	129
	NS		140	113	113	113	113	113
	KL	2	135	101	101	101	101	101
StartSchedule			59	59	59	59	59	59
StopSchedule			36	36	36	36	36	36
GetScheduleStatus			72	72	72	72	72	72
GetScheduleValue			56	56	56	56	56	56
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			32	32	32	32	32	32
SetArrivalpointDelay			29	29	29	29	29	29
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			96	96	96	96	96	96
GetLargestExecutionTime			36	36	36	36	36	36
ResetLargestExecutionTime			30	30	30	30	30	30
GetStackOffset			18	18	18	18	18	18

Extended

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	144	193	231	150	199	255
	NS		185	236	273	191	242	297
	KL	2	124	175	213	130	181	237
TerminateTask	LExt	3	77	77	77	77	77	77

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	103	103	103	103	103	103
ChainTask	SWL	1, 8	168	217	254	174	223	278
	SWH	1, 9	200	247	285	206	253	309
	NSL	8	222	277	314	231	283	338
	NSH	9	246	298	336	255	305	361
Schedule			137	137	154	137	137	154
GetTaskID			32	32	32	32	32	32
GetTaskState			146	146	146	152	152	152
EnableAllInterrupts			17	17	17	17	17	17
DisableAllInterrupts			23	23	23	23	23	23
ResumeAllInterrupts			51	51	51	51	51	51
SuspendAllInterrupts			33	33	33	33	33	33
ResumeOSInterrupts			51	51	51	51	51	51
SuspendOSInterrupts			39	39	39	39	39	39
GetResource	Task	7	240	240	209	240	240	209
	Combined	6	213	213	213	213	213	213
	CLEx	3	207	207	207	207	207	207
ReleaseResource	Task	7	213	213	213	213	213	213
	Combined	6	260	260	260	260	260	260
	CLEx	3	190	190	190	190	190	190
SetEvent	SW	1	n/a	n/a	n/a	189	189	262
	NS		n/a	n/a	n/a	230	230	302
	NS1i	10	n/a	n/a	n/a	135	n/a	n/a
	KL	2	n/a	n/a	n/a	184	184	244
	KL1i	2, 10	n/a	n/a	n/a	147	n/a	n/a
ClearEvent			n/a	n/a	n/a	96	96	96
GetEvent			n/a	n/a	n/a	127	127	127
WaitEvent	<default>		n/a	n/a	n/a	292	292	443
	fp	11	n/a	n/a	n/a	316	316	494
	1i	10	n/a	n/a	n/a	142	n/a	n/a
GetAlarmBase			94	94	94	94	94	94
GetAlarm			113	113	113	113	113	113
SetRelAlarm			923	923	923	923	923	923
SetAbsAlarm			1056	1056	1056	1056	1056	1056
CancelAlarm			86	86	86	86	86	86
InitCounter			170	170	170	170	170	170
GetCounterValue			140	140	140	140	140	140

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	64	90	90	64	90	90
NextScheduleTable		34	82	204	204	82	204	204
StartScheduleTable		34	110	157	157	110	157	157
StopScheduleTable		34	77	95	95	77	95	95
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		5	5	5	5	5	5
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		27	27	27	27	27	27
GetISRID		4	44	44	44	44	44	44
Process container	Yielding	32	25	25	25	25	25	25
Process container	Non-Yielding	33	13	13	13	13	13	13
osek_tick_alarm	<default>		76	76	76	76	76	76
	KL	2	59	59	59	59	59	59
osek_incr_counter			74	74	74	74	74	74
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			140	140	140	140	140	140
ShutdownOS	NoHook	12	16	16	16	16	16	16
	Hook	13	23	23	23	23	23	23
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			24	24	24	24	24	24
StopCOM			24	24	24	24	24	24
ReadFlag			18	18	18	18	18	18
ResetFlag			19	19	19	19	19	19
ReceiveMessage	CCCA	14	80	80	80	169	169	169
	CCCB	15	169	169	169	169	169	169
GetMessageResource			60	60	60	60	60	60
ReleaseMessageResource			60	60	60	60	60	60
GetMessageStatus			68	68	68	68	68	68
SendMessage	SW CCCA	1, 14	119	119	119	232	232	232
	SW CCCB	1, 15	211	211	211	232	232	232
	NS CCCA	14	119	119	119	232	232	232
	NS CCCB	15	211	211	211	232	232	232
	KL CCCA	2, 14	111	111	111	245	245	245
	KL CCCB	2, 15	219	219	219	245	245	245
main_dispatch	NoHook	12	184	184	224	184	184	224

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	215	215	255	215	215	255
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	64	64	64	64	64	64
	B1HF	21	74	74	74	74	74	74
	B2LI	22	n/a	33	62	n/a	33	62
	B2LF	23	n/a	41	70	n/a	41	70
	B2HI	24	n/a	146	212	n/a	146	212
	B2HF	25	n/a	154	220	n/a	154	220
	E1HI	26	n/a	n/a	n/a	262	262	334
	E1HF	27	n/a	n/a	n/a	270	270	342
	E2HI	28	n/a	n/a	n/a	n/a	n/a	334
	E2HF	29	n/a	n/a	n/a	n/a	n/a	342
ErrorHook support		16	40	40	40	40	40	40
	ServiceID	17	45	45	45	45	45	45
	Parameters	18	76	76	76	76	76	76
validity_checks		3	26	26	26	26	26	26
Timing_dispatch		4	45	45	45	45	45	45
Timing_termination		4	85	85	85	85	85	85
ActivateTaskset	SW	1	193	246	296	207	274	332
	NS		234	286	336	248	313	372
	KL	2	164	217	265	178	245	306
ChainTaskset	SWL	1, 8	241	293	342	251	314	374
	SWH	1, 9	280	343	393	290	366	427
	NSL	8	293	345	395	300	363	423
	NSH	9	326	390	442	333	409	470
GetTasksetRef			98	98	98	98	98	98
MergeTaskset			139	139	139	139	139	139
AssignTaskset			119	119	119	119	119	119
RemoveTaskset			141	141	141	141	141	141
TestSubTaskset			152	152	152	152	152	152
TestEquivalentTaskset			145	145	145	145	145	145
TickSchedule	SW	1	262	222	222	222	222	222
	NS		307	297	297	297	297	297
	KL	2	262	220	220	220	220	220
AdvanceSchedule	SW	1	236	201	201	201	201	201
	NS		281	273	273	273	273	273
	KL	2	242	197	197	197	197	197

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			153	153	153	153	153	153
StopSchedule			89	89	89	89	89	89
GetScheduleStatus			126	126	126	126	126	126
GetScheduleValue			105	105	105	105	105	105
GetScheduleNext			61	61	61	61	61	61
SetScheduleNext			113	113	113	113	113	113
GetArrivalpointDelay			104	104	104	104	104	104
SetArrivalpointDelay			130	130	130	130	130	130
GetArrivalpointTasksetRef			84	84	84	84	84	84
GetArrivalpointNext			88	88	88	88	88	88
SetArrivalpointNext			127	127	127	127	127	127
TestArrivalpointWritable			98	98	98	98	98	98
GetExecutionTime			106	106	106	106	106	106
GetLargestExecutionTime			105	105	105	105	105	105
ResetLargestExecutionTime			92	92	92	92	92	92
GetStackOffset			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.
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

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	104	139	195	108	147	209
	NS	94	129	185	98	137	199
	KL	67	91	151	71	99	165
TerminateTask	LExt	0	0	0	0	0	0
	H	140	138	145	138	138	145
ChainTask	SWL	232	268	369	279	323	419
	SWH	292	324	424	338	378	475
	NSL	232	268	369	279	323	419
	NSH	287	319	419	333	373	470
Schedule	SW	83	81	93	81	81	93
GetTaskID		36	35	35	35	35	35
GetTaskState		94	93	94	105	106	105
EnableAllInterrupts		27	27	27	27	27	27
DisableAllInterrupts		43	43	43	43	43	43
ResumeAllInterrupts		35	35	35	35	35	35
SuspendAllInterrupts		42	42	42	42	42	42
ResumeOSInterrupts		35	35	35	35	35	35
SuspendOSInterrupts		42	42	42	42	42	42
GetResource	Task	46	46	58	43	54	55
	Combined	53	53	53	50	50	50
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	74	74	74	72	72	72
	Combined	67	67	67	64	64	64
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	115	115	115
	NS	n/a	n/a	n/a	115	115	115
	KL	n/a	n/a	n/a	91	91	91
ClearEvent		n/a	n/a	n/a	44	44	44
GetEvent		n/a	n/a	n/a	37	37	37
WaitEvent	<default>	n/a	n/a	n/a	325	336	402
	fp	n/a	n/a	n/a	334	346	411
GetAlarmBase		176	175	175	187	176	176
GetAlarm		114	111	111	114	114	114

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		150	145	145	150	150	150
SetAbsAlarm		146	141	141	146	146	146
CancelAlarm		70	69	69	70	70	70
InitCounter		94	91	91	94	94	94
GetCounterValue		96	93	93	96	96	96
osek_tick_alarm	<default>	125	124	124	125	125	125
	KL	90	89	89	90	90	90
osek_incr_counter		24	23	23	24	24	24
GetActiveApplicationMode		17	17	6	17	17	6
StartOS		468	467	467	467	467	467
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	33	33	33	33	33	33
InitCOM		29	18	18	18	29	29
CloseCOM		18	18	18	18	18	18
StartCOM		54	54	54	202	202	202
StopCOM		26	26	26	26	26	26
ReadFlag		n/a	n/a	n/a	14	14	14
ResetFlag		n/a	n/a	n/a	10	10	10
ReceiveMessage		70	72	72	218	218	218
GetMessageResource		n/a	n/a	n/a	97	97	97
ReleaseMessageResource		n/a	n/a	n/a	114	114	114
GetMessageStatus		n/a	n/a	n/a	52	52	52
SendMessage	SW	211	248	304	368	407	469
	NS	198	235	302	355	394	456
	KL	126	163	223	314	353	419
ActivateTaskset	SW	80	473	532	99	482	556
	NS	67	449	519	75	458	532
	KL	35	343	413	43	430	427
	SW2	80	462	532	88	471	545
	NS2	67	449	519	75	458	532
	KL2	35	343	413	43	430	427
ChainTaskset	SWL	231	601	716	262	652	757
	SWH	278	659	774	319	710	815
	NSL	220	601	716	262	652	757
	NSH	273	654	769	314	705	810
GetTasksetRef		34	33	34	33	34	33

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		70	70	70	70	70	70
AssignTaskset		32	32	32	32	32	32
RemoveTaskset		72	72	72	72	72	72
TestSubTaskset		85	85	85	85	85	85
TestEquivalentTaskset		76	76	76	76	76	76
TickSchedule	SW	214	552	626	262	660	662
	NS	192	536	599	235	633	635
	KL	187	524	587	223	621	623
	SW2	214	551	621	251	638	635
	NS2	192	535	605	235	622	619
	KL2	187	523	593	223	610	607
AdvanceSchedule	SW	151	498	561	197	595	597
	NS	135	482	545	181	579	581
	KL	119	456	519	155	553	555
	SW2	151	497	567	197	584	581
	NS2	135	481	551	181	568	565
	KL2	119	455	525	155	542	539
StartSchedule		103	103	103	103	103	103
StopSchedule		78	78	89	78	78	78
GetScheduleStatus		102	102	102	102	102	102
GetScheduleValue		90	90	90	90	90	90
GetScheduleNext		36	36	36	36	36	36
SetScheduleNext		33	33	33	33	33	33
GetArrivalpointDelay		72	72	72	72	72	72
SetArrivalpointDelay		72	72	72	72	72	72
GetArrivalpointTasksetRef		29	29	29	29	29	29
GetArrivalpointNext		32	32	32	32	32	32
SetArrivalpointNext		29	29	29	29	29	29
TestArrivalpointWritable		38	38	38	38	38	38
GetExecutionTime		17	17	17	17	17	17
GetLargestExecutionTime		35	34	34	34	34	34
ResetLargestExecutionTime		19	18	18	18	18	18
GetStackOffset		34	34	34	34	34	34

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	104	139	195	108	147	209
	NS	94	129	185	98	137	199
	KL	56	91	151	60	99	165
TerminateTask	LExt	0	0	0	0	0	0
	H	569	564	571	564	564	571
ChainTask	SWL	719	750	865	763	808	916
	SWH	775	802	916	818	859	968
	NSL	719	750	865	763	808	916
	NSH	751	779	893	795	836	945
Schedule	SW	85	83	95	83	83	95
GetTaskID		36	46	35	35	46	35
GetTaskState		94	93	94	105	106	105
EnableAllInterrupts		27	27	27	27	27	27
DisableAllInterrupts		43	32	43	32	43	43
ResumeAllInterrupts		35	35	35	35	35	35
SuspendAllInterrupts		42	42	42	42	42	42
ResumeOSInterrupts		35	35	35	35	35	35
SuspendOSInterrupts		42	42	42	42	42	42
GetResource	Task	57	57	47	54	43	44
	Combined	53	53	53	50	50	50
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	74	74	74	72	72	72
	Combined	67	67	67	64	64	64
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	115	115	115
	NS	n/a	n/a	n/a	115	115	115
	KL	n/a	n/a	n/a	91	91	91
ClearEvent		n/a	n/a	n/a	44	44	44
GetEvent		n/a	n/a	n/a	37	37	37
WaitEvent	<default>	n/a	n/a	n/a	748	762	827
	fp	n/a	n/a	n/a	757	772	836
GetAlarmBase		176	175	175	176	176	176
GetAlarm		114	111	111	114	114	114

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		150	145	145	150	150	150
SetAbsAlarm		146	141	141	146	146	146
CancelAlarm		70	69	69	70	70	70
InitCounter		94	91	91	94	94	94
GetCounterValue		96	93	93	96	96	96
osek_tick_alarm	<default>	125	124	124	125	125	125
	KL	90	89	89	90	90	90
osek_incr_counter		24	23	23	24	24	24
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		1115	1114	1114	1113	1113	1113
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	33	33	33	33	33	33
InitCOM		18	18	18	18	18	18
CloseCOM		18	18	18	18	18	18
StartCOM		54	54	54	202	202	202
StopCOM		26	26	26	26	26	26
ReadFlag		n/a	n/a	n/a	14	14	14
ResetFlag		n/a	n/a	n/a	10	10	10
ReceiveMessage		70	72	72	218	218	218
GetMessageResource		n/a	n/a	n/a	98	98	98
ReleaseMessageResource		n/a	n/a	n/a	114	114	114
GetMessageStatus		n/a	n/a	n/a	52	52	52
SendMessage	SW	211	248	304	368	407	469
	NS	198	235	291	355	394	456
	KL	126	163	223	314	353	419
ActivateTaskset	SW	80	462	532	88	471	545
	NS	67	449	519	75	458	532
	KL	35	343	413	43	441	427
	SW2	80	462	532	88	471	545
	NS2	67	449	519	75	458	532
	KL2	35	343	413	43	430	427
ChainTaskset	SWL	689	1083	1212	729	1137	1254
	SWH	742	1137	1266	781	1191	1308
	NSL	689	1083	1212	729	1137	1254
	NSH	737	1132	1243	776	1168	1285
GetTasksetRef		34	33	34	33	34	33

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		70	70	70	70	70	70
AssignTaskset		32	32	32	32	32	32
RemoveTaskset		72	72	72	72	72	72
TestSubTaskset		85	85	85	85	85	85
TestEquivalentTaskset		76	76	76	76	76	76
TickSchedule	SW	214	552	615	251	649	651
	NS	192	536	599	235	633	635
	KL	187	524	587	223	621	623
	SW2	214	551	621	251	638	635
	NS2	192	535	605	235	622	619
	KL2	187	523	593	223	610	607
AdvanceSchedule	SW	151	498	561	197	595	597
	NS	135	482	545	181	579	581
	KL	119	456	519	155	553	555
	SW2	151	497	567	197	584	581
	NS2	135	481	551	181	568	565
	KL2	119	455	525	155	542	539
StartSchedule		103	103	103	103	103	103
StopSchedule		78	78	78	78	78	78
GetScheduleStatus		102	102	102	102	102	102
GetScheduleValue		90	90	90	90	90	90
GetScheduleNext		36	36	36	36	36	36
SetScheduleNext		33	33	33	33	33	33
GetArrivalpointDelay		72	72	72	72	72	72
SetArrivalpointDelay		72	72	72	72	72	72
GetArrivalpointTasksetRef		29	29	29	29	29	29
GetArrivalpointNext		32	32	32	32	32	32
SetArrivalpointNext		29	29	29	29	29	29
TestArrivalpointWritable		38	38	38	38	38	38
GetExecutionTime		248	248	248	247	247	247
GetLargestExecutionTime		81	79	79	79	79	79
ResetLargestExecutionTime		66	64	64	64	64	64
GetStackOffset		34	34	34	34	34	34

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	351	363	419	329	371	433
	NS	388	398	454	364	407	467
	KL	324	336	392	302	344	406
TerminateTask	LExt	590	584	591	584	584	591
	H	634	627	634	627	627	634
ChainTask	SWL	1014	1020	1135	1030	1078	1186
	SWH	1070	1072	1186	1085	1129	1238
	NSL	1059	1065	1180	1075	1124	1230
	NSH	1109	1111	1225	1124	1169	1276
Schedule	SW	110	107	119	107	107	119
GetTaskID		55	54	54	54	54	54
GetTaskState		368	341	342	346	347	346
EnableAllInterrupts		35	35	35	35	35	35
DisableAllInterrupts		51	51	51	51	51	51
ResumeAllInterrupts		49	49	49	49	49	49
SuspendAllInterrupts		50	50	50	50	50	50
ResumeOSInterrupts		49	49	49	49	49	49
SuspendOSInterrupts		50	50	50	50	50	50
GetResource	Task	743	710	309	715	718	330
	Combined	261	261	261	285	285	285
	CLEx	326	322	322	348	348	348
ReleaseResource	Task	281	281	281	305	305	305
	Combined	251	251	251	276	276	276
	CLEx	287	284	284	311	311	311
SetEvent	SW	n/a	n/a	n/a	353	353	353
	NS	n/a	n/a	n/a	375	375	375
	KL	n/a	n/a	n/a	355	355	344
ClearEvent		n/a	n/a	n/a	80	80	80
GetEvent		n/a	n/a	n/a	306	306	306
WaitEvent	<default>	n/a	n/a	n/a	801	816	884
	fp	n/a	n/a	n/a	810	826	882
GetAlarmBase		326	312	326	326	326	342
GetAlarm		264	248	262	264	264	280

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		365	340	365	365	365	381
SetAbsAlarm		337	315	329	337	337	353
CancelAlarm		220	206	220	220	220	236
InitCounter		452	417	445	452	452	483
GetCounterValue		230	216	216	230	230	230
osek_tick_alarm	<default>	125	124	124	125	125	125
	KL	90	89	89	90	90	90
osek_incr_counter		24	23	23	24	24	24
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		1139	1138	1138	1137	1137	1137
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	37	37	37	37	37	37
InitCOM		18	18	18	18	18	18
CloseCOM		18	18	18	18	18	18
StartCOM		66	66	66	214	214	214
StopCOM		37	37	37	37	37	37
ReadFlag		n/a	n/a	n/a	37	37	37
ResetFlag		n/a	n/a	n/a	34	34	34
ReceiveMessage		171	180	180	319	319	319
GetMessageResource		n/a	n/a	n/a	472	472	472
ReleaseMessageResource		n/a	n/a	n/a	446	446	446
GetMessageStatus		n/a	n/a	n/a	152	152	152
SendMessage	SW	559	580	636	690	732	794
	NS	593	612	668	722	765	825
	KL	521	542	598	675	717	779
ActivateTaskset	SW	552	951	1022	563	959	1038
	NS	581	979	1050	591	987	1066
	KL	391	795	793	402	798	801
	SW2	552	951	1022	563	959	1038
	NS2	581	979	1050	591	987	1066
	KL2	391	795	793	402	798	801
ChainTaskset	SWL	1253	1646	1776	1300	1699	1824
	SWH	1308	1699	1829	1354	1752	1889
	NSL	1295	1687	1817	1339	1738	1863
	NSH	1344	1734	1865	1387	1785	1911
GetTasksetRef		302	275	276	275	276	275

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		116	116	116	116	116	116
AssignTaskset		84	84	84	84	84	84
RemoveTaskset		116	116	116	116	116	116
TestSubTaskset		131	131	131	131	131	131
TestEquivalentTaskset		122	122	122	122	122	122
TickSchedule	SW	255	1020	1009	625	1052	1065
	NS	293	1060	1049	665	1092	1105
	KL	248	1012	1001	617	1044	1057
	SW2	255	1018	1016	625	1021	1024
	NS2	293	1058	1056	665	1061	1064
	KL2	248	1010	1008	617	1013	1016
AdvanceSchedule	SW	204	976	965	581	1008	1021
	NS	243	1010	999	615	1042	1055
	KL	199	957	946	562	989	1002
	SW2	204	974	972	581	977	980
	NS2	243	1008	1006	615	1011	1014
	KL2	199	955	953	562	958	961
StartSchedule		139	139	139	139	139	139
StopSchedule		94	94	94	94	94	94
GetScheduleStatus		118	118	118	118	118	118
GetScheduleValue		106	106	106	106	106	106
GetScheduleNext		58	58	58	58	58	58
SetScheduleNext		84	84	84	84	84	84
GetArrivalpointDelay		97	97	97	97	97	97
SetArrivalpointDelay		110	110	110	110	110	110
GetArrivalpointTasksetRef		52	52	52	52	52	52
GetArrivalpointNext		56	56	56	56	56	56
SetArrivalpointNext		89	89	89	89	89	89
TestArrivalpointWritable		61	61	61	61	61	61
GetExecutionTime		256	267	256	266	255	266
GetLargestExecutionTime		327	300	300	300	300	300
ResetLargestExecutionTime		312	285	285	285	285	285
GetStackOffset		34	34	34	34	34	34

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	22	22	22	22	22	22
	Cat 2	31	59	59	59	59	59

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	22	22	22	22	22	22
	Cat 2	391	412	410	410	410	410

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	22	22	22	22	22	22
	Cat 2	391	412	412	410	408	410

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.

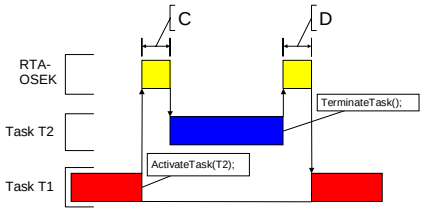

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

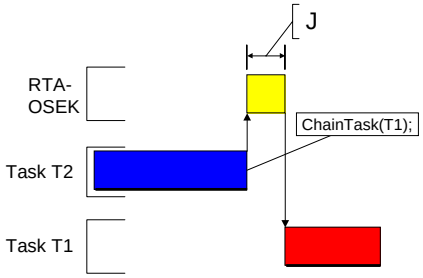

Figure 2: Task Chaining

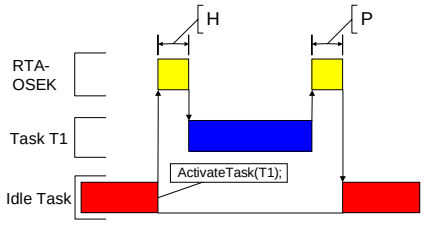

Figure 3: Task Activation from Idle Task

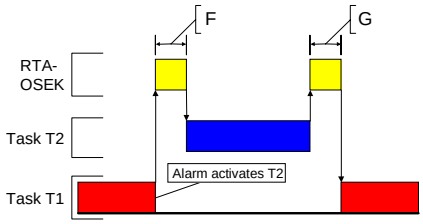

Figure 4: Task Activation from an Alarm

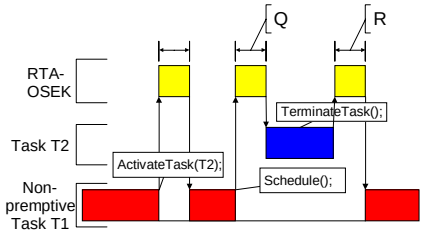

Figure 5: Non-Premptive Task Calls Schedule()

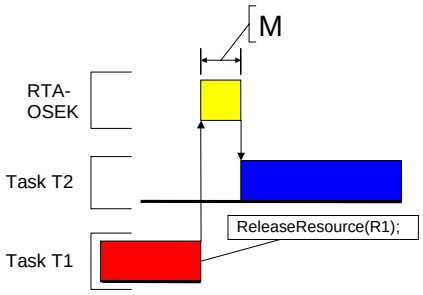

Figure 6: Blocked Task Activated by ReleaseResource()

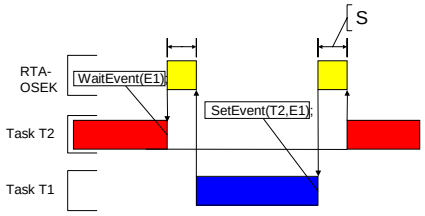

Figure 7: Waiting Task Activated by SetEvent()

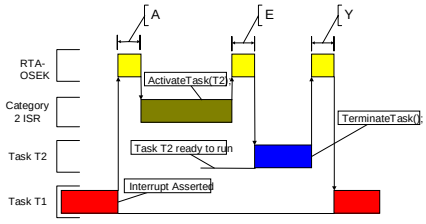

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	75	99	137	74	99	142
Figure 1: D	Heavy, Basic/Extended	140	154	198	167	171	205
ChainTask	Light, Basic	177	227	321	176	221	329
Figure 2: J	Heavy, Basic/Extended	490	549	687	515	561	703
Pre-emption	Light, Basic	161	213	302	162	213	334
Figure 1: C	Heavy, Basic/Extended	220	257	365	268	312	415
From idle task	Light, Basic	161	213	302	162	213	334
Figure 3: H	Heavy, Basic/Extended	220	257	365	268	312	415
Triggered by alarm	Light, Basic	320	371	460	320	371	492
Figure 4: F	Heavy, Basic/Extended	379	415	523	426	470	573
Schedule	Light, Basic	130	141	201	126	142	205
Figure 5: Q	Heavy, Basic/Extended	189	190	254	232	238	291
Release resource	Light, Basic	143	155	203	137	152	203
Figure 6: M	Heavy, Basic/Extended	202	204	256	243	248	289
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	405	419	554
From category 2 ISR	Light, Basic	110	150	198	135	150	201
Figure 8: E	Heavy, Basic/Extended	169	199	251	241	246	287

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	511	534	567	507	534	575
Figure 1: D	Heavy, Basic/Extended	567	577	626	590	597	630
ChainTask	Light, Basic	670	710	817	665	686	807
Figure 2: J	Heavy, Basic/Extended	1424	1468	1626	1440	1465	1619
Pre-emption	Light, Basic	529	575	676	528	573	707
Figure 1: C	Heavy, Basic/Extended	582	618	740	629	674	789
From idle task	Light, Basic	508	554	655	507	552	686
Figure 3: H	Heavy, Basic/Extended	561	597	719	608	653	768
Triggered by alarm	Light, Basic	688	733	834	686	731	865
Figure 4: F	Heavy, Basic/Extended	741	776	898	787	832	947
Schedule	Light, Basic	502	505	577	496	508	584
Figure 5: Q	Heavy, Basic/Extended	555	553	631	597	606	671
Release resource	Light, Basic	511	517	577	503	512	576
Figure 6: M	Heavy, Basic/Extended	564	565	631	604	610	663
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	744	758	906
From category 2 ISR	Light, Basic	917	941	1001	930	939	1003
Figure 8: E	Heavy, Basic/Extended	970	989	1055	1031	1037	1090

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	592	614	646	586	614	655
Figure 1: D	Heavy, Basic/Extended	632	640	690	653	661	693
ChainTask	Light, Basic	965	980	1087	932	956	1077
Figure 2: J	Heavy, Basic/Extended	1784	1801	1960	1770	1799	1952
Pre-emption	Light, Basic	772	795	896	745	793	927
Figure 1: C	Heavy, Basic/Extended	825	838	960	846	894	1009
From idle task	Light, Basic	751	774	875	724	772	906
Figure 3: H	Heavy, Basic/Extended	804	817	939	825	873	988
Triggered by alarm	Light, Basic	931	953	1054	903	951	1085
Figure 4: F	Heavy, Basic/Extended	984	996	1118	1004	1052	1167
Schedule	Light, Basic	523	525	597	516	529	605
Figure 5: Q	Heavy, Basic/Extended	576	573	651	617	627	692
Release resource	Light, Basic	707	713	773	726	735	799
Figure 6: M	Heavy, Basic/Extended	760	761	827	827	833	886
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	981	995	1143
From category 2 ISR	Light, Basic	925	949	1009	938	947	1011
Figure 8: E	Heavy, Basic/Extended	978	997	1063	1039	1045	1098

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	20	20	23	20	20	23
BCC1 lightweight, floating-point	23	23	26	23	23	26
BCC1 heavyweight, integer	28	28	31	28	28	31
BCC1 heavyweight, floating-point	28	28	31	28	28	31
BCC2 lightweight, integer	n/a	23	28	n/a	23	28
BCC2 lightweight, floating-point	n/a	23	28	n/a	23	28
BCC2 heavyweight, integer	n/a	32	39	n/a	32	39
BCC2 heavyweight, floating-point	n/a	32	39	n/a	32	39
ECC1 heavyweight, integer	n/a	n/a	n/a	38	38	41
ECC1 heavyweight, floating-point	n/a	n/a	n/a	38	38	41
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	45
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	45
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	23	23	23	23	23	23
BCC1 lightweight, floating-point	26	26	26	26	26	26
BCC1 heavyweight, integer	31	31	31	31	31	31
BCC1 heavyweight, floating-point	31	31	31	31	31	31
BCC2 lightweight, integer	n/a	26	28	n/a	26	28
BCC2 lightweight, floating-point	n/a	26	28	n/a	26	28
BCC2 heavyweight, integer	n/a	35	39	n/a	35	39
BCC2 heavyweight, floating-point	n/a	35	39	n/a	35	39
ECC1 heavyweight, integer	n/a	n/a	n/a	41	41	41
ECC1 heavyweight, floating-point	n/a	n/a	n/a	41	41	41
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	45
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	45

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	29	29	32	29	29	32
BCC1 lightweight, floating-point	32	32	35	32	32	35
BCC1 heavyweight, integer	37	37	40	37	37	40
BCC1 heavyweight, floating-point	37	37	40	37	37	40
BCC2 lightweight, integer	n/a	32	37	n/a	32	37
BCC2 lightweight, floating-point	n/a	32	37	n/a	32	37
BCC2 heavyweight, integer	n/a	41	48	n/a	41	48
BCC2 heavyweight, floating-point	n/a	41	48	n/a	41	48
ECC1 heavyweight, integer	n/a	n/a	n/a	47	47	50
ECC1 heavyweight, floating-point	n/a	n/a	n/a	47	47	50
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	54
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	54
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	32	32	32	32	32	32
BCC1 lightweight, floating-point	35	35	35	35	35	35
BCC1 heavyweight, integer	40	40	40	40	40	40
BCC1 heavyweight, floating-point	40	40	40	40	40	40
BCC2 lightweight, integer	n/a	35	37	n/a	35	37
BCC2 lightweight, floating-point	n/a	35	37	n/a	35	37
BCC2 heavyweight, integer	n/a	44	48	n/a	44	48
BCC2 heavyweight, floating-point	n/a	44	48	n/a	44	48
ECC1 heavyweight, integer	n/a	n/a	n/a	50	50	50
ECC1 heavyweight, floating-point	n/a	n/a	n/a	50	50	50
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	54
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	54

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	29	29	32	29	29	32
BCC1 lightweight, floating-point	32	32	35	32	32	35
BCC1 heavyweight, integer	37	37	40	37	37	40
BCC1 heavyweight, floating-point	37	37	40	37	37	40
BCC2 lightweight, integer	n/a	32	37	n/a	32	37
BCC2 lightweight, floating-point	n/a	32	37	n/a	32	37
BCC2 heavyweight, integer	n/a	41	48	n/a	41	48
BCC2 heavyweight, floating-point	n/a	41	48	n/a	41	48
ECC1 heavyweight, integer	n/a	n/a	n/a	47	47	50
ECC1 heavyweight, floating-point	n/a	n/a	n/a	47	47	50
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	54
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	54
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	32	32	32	32	32	32
BCC1 lightweight, floating-point	35	35	35	35	35	35
BCC1 heavyweight, integer	40	40	40	40	40	40
BCC1 heavyweight, floating-point	40	40	40	40	40	40
BCC2 lightweight, integer	n/a	35	37	n/a	35	37
BCC2 lightweight, floating-point	n/a	35	37	n/a	35	37
BCC2 heavyweight, integer	n/a	44	48	n/a	44	48
BCC2 heavyweight, floating-point	n/a	44	48	n/a	44	48
ECC1 heavyweight, integer	n/a	n/a	n/a	50	50	50
ECC1 heavyweight, floating-point	n/a	n/a	n/a	50	50	50
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	54
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	54

5 Compatibility with Pre-v5 Kernels

5.1 Updating the Application Version

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

5.2 Memory Model

The v3.x kernel was built using the small memory model, whereas the v5.0 kernel was built using the **banked** memory model. Existing applications should be modified so that application code is compiled or assembled to use the banked memory model.

5.3 32 Bit Timer Drivers

The v3.x kernel uses sixteen bit timer values, whereas the v5.0 kernel uses thirty-two bit timer values. Therefore any existing applications' timer drivers will need modifying. Since the HC(S)12 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 `TCNT` timer register.

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.