RTA-OSEK

Binding Manual: TriCore/Tasking

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

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

www.etas.de

Japan

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

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

www.etas.co.jp

Korea

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

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

www.etas.co.kr

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

France

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

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

www.etas.fr

Great Britain

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

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

www.etas-uk.net

Copyright Notice

Version: RM00058-004

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 TriCore/Tasking 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 Compiler

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

Vendor	Tasking
Compiler	TriCore VX-toolset C compiler
Version	v2.1r1 Build 116.1.3

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

Option	**Description**
`-C<cpu>`	Selects target CPU
`--silicon-bug=all-tc113`	Includes workarounds for v1.3 CPU silicon bugs

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**
`-C<cpu>`	Selects target CPU
`--silicon-bug=all-tc113`	Includes workarounds for v1.3 CPU silicon bugs

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

Option	**Description**
`-g`	Enable debugging information

The startup code supplied with the Tasking toolset is fully compatible with RTA-OSEK and does not require modification.

The compiler is used with flags to include workarounds for known TriCore silicon problems.

Important: If tracing is enabled then the compiler will generate the following warnings when compiling the automatically generated file `osekdefs.c`:

Warning W517 “ambiguous 'else' part - suggest braces” - The code to which this warning refers compiles in the correct way.

Warning W549 “condition is constant” - The use of a constant condition is intentional and allows the compiler to optimize away some unneeded code.

Warning W560 “possible truncation at implicit conversion to type "xxxx"” - This warning arises because RTA-TRACE uses different sized data items depending on the tracing mode (simple or advanced) and on the target processor. The code to which the warning refers compiles in the correct way.

The above warnings are benign and build scripts generated by RTA-OSEK automatically suppress these warnings.

2.2 Assembler

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

Vendor	Tasking
Assembler	TriCore VX-toolset assembler
Version	v2.1r1 Build 106

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

Option	Description
`-C<cpu>`	Selects target CPU
`--silicon-bug=all-tc113`	Includes workarounds for v1.3 CPU silicon bugs

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
`-C<cpu>`	Selects target CPU
`--silicon-bug=all-tc113`	Includes workarounds for v1.3 CPU silicon bugs

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

Option	Description
`-g`	Enable debugging information

Important: The assembler will generate warning W292 "suspicious instruction concerning CPU functional defect TC113_CPU14" when assembling the automatically generated file `osgen.asm`. Functional defect TC113_CPU14 is not present in any production TriCore v1.3 processor cores and so the warning may be ignored (the defect was present in some prototype v1.3 processor cores). Build scripts generated by RTA-OSEK automatically suppress this warning.

2.3 Linker/Locator

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

Sections	Rom/Ram	Description
`os_pid`	ROM	RTA-OSEK read-only data
`os_pird`	ROM	RTA-OSEK initialization data
`os_pnird`	ROM	RTA-OSEK near initialization data
`os_pir`	RAM	RTA-OSEK initialized data
`os_pur`	RAM	RTA-OSEK uninitialized data
`os_pnir`	RAM	RTA-OSEK near initialized data

The RTA-OSEK Component requires the user stack to be quad word aligned. In order to achieve this, `__TC112_COR16__` should be defined when using the linker script files supplied with the Tasking toolchain.

Important: The RTA-OSEK Component makes use of relative 24-bit signed addressing mode. This means the library must be contained within a 1024K byte memory block. 32-bit addressing is used externally providing no restrictions on placement of user code and data.

2.4 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	Tasking Crossview Pro 2.1r1 Build 053 OSEK/ORTI v2.0, RADM 009

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 *Infineon TriCore User's Manual - System Units*.

The following table shows how RTA-OSEK IPLs relate to interrupt priorities on the target hardware (see section 3.1.7 for an explanation of the relationship between interrupt priorities and vectors):

IPL Value	CCPN Field Of ICR Register	Description
0	0	User level
1-255	1-255	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 (see section 3.1.7 for an explanation of the relationship between interrupt priorities and vectors):

Vector	Legality
1-255	Category 1 Note all Category 1 interrupt vectors must be configured to be higher than the highest Category 2 interrupt vector.
1-255	Category 2

The valid base addresses for the vector table are:

Base Address	Notes
BIV	Set by this register. See TriCore Architecture manual for a full explanation.

3.1.3 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Tasking C compiler can generate appropriate interrupt handling code for a C function decorated with the `__interrupt(<vector>)` `__enable_` 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

RTA-OSEK relies on the compiler-generated vector table and does not produce its own table. For this reason, the Vector Generation option in the RTA-OSEK GUI has no effect.

The following table shows the syntax for labels attached to RTA-OSEK Category 2 interrupt handlers.

Vector Location	Label
`BIV + 0x0020`	`os_wrapper_001`
`BIV + 0x0040`	`os_wrapper_002`
...	...
`BIV + 0x1fe0`	`os_wrapper_0ff`

3.1.6 IO Privilege Mode

The RTA-OSEK Component operates with the TriCore CPU in supervisor mode at all times and this must not be altered.

3.1.7 Interrupt Priorities

When an interrupt becomes pending, it is handled as soon as the configured vector number is strictly greater than the current hardware priority value in ICR.CCPN.

RTA-OSEK supports a straightforward, pre-emptive interrupt model, where each ISR runs at the same priority as the vector number. This matches the default TriCore interrupt behavior. To achieve this, configure the Priority for each ISR to be the same as the Vector for that ISR.

For example, an application with two Category 2 ISRs, A and B and two Category 1 ISRs, C and D, could be configured as follows:

ISR	Category	Vector	Priority
D	1	4	4
C	1	3	3
B	2	2	2
A	2	1	1

Note that the range of interrupt vector used does not need to be contiguous – i.e. there can be gaps in the range of vectors used.

The Category 1 ISRs would be written as

```
void __interrupt(4) __enable_ D(void)
{
  /* handler code for D */
}
```

and

```
void __interrupt(3) __enable_ C(void)
{
  /* handler code for C */
}
```

Serialized Category 2 ISRs

In addition to the straightforward, pre-emptive interrupt model, RTA-OSEK also supports serialized Category 2 ISRs. A contiguous group of Category 2 ISRs, with lowest Vector `u` and highest Vector `v`, can be serialized by raising all their Priorities to `v`.

The RTA-OSEK Component starts Category 2 ISRs at their specified Priority, achieving the appropriate serialization at run-time.

For correct schedulability analysis in the presence of serialized, or "shared priority" Category 2 ISRs, interrupt arbitration information must be entered for each shared priority. The arbitration order must be set so that the higher vector numbers come earlier in the arbitration order.

For example, an application with two Category 2 ISRs, A and B and two Category 1 ISRs, C and D, could be configured as follows:

ISR	Category	Vector	Priority
D	1	4	4
C	1	3	3
B	2	2	2
A	2	1	2
Arbitration	Level 2 ordering: B, A.		

Serialization causes higher vector ISRs to be delayed until the completion of lower vector ISRs of the same priority but it gives the following advantages:

- If a resource is used only by a group of serialized ISRs, they can get and release that resource at zero overhead, using the RTA-OSEK static interface to resources.
- Inhibiting pre-emption between ISRs reduces the worst-case stack and CSA requirements for an application.

Serialized Category 1 ISRs

RTA-OSEK also supports serialization of Category 1 ISRs. The highest vector Category 1 ISRs can be serialized by raising all their Priorities to `255`.

Furthermore, such serialized Category 1 ISRs must be written using either the `__interrupt(<vector>)` or `__interrupt_fast(<vector>)` function qualifiers *without* the `__enable_` modifier and they are not permitted to use the `__enable()` intrinsic. This ensures that they execute with interrupts disabled, giving the appropriate serialization at run-time.

For correct schedulability analysis in the presence of serialized, or "shared priority" Category 1 ISRs, interrupt arbitration information must be entered for priority `255`. The arbitration order must be set so that the higher vector numbers come earlier in the arbitration order.

For example, an application with two Category 2 ISRs, A and B and two Category 1 ISRs, C and D, could be configured as follows:

ISR	Category	Vector	Priority
D	1	4	255
C	1	3	255
B	2	2	2
A	2	1	2
Arbitration	Level 255 ordering: D, C.		
Arbitration	Level 2 ordering: B, A.		

The Category 1 ISRs would be written as

```
void __interrupt(4) D(void)
{
  /* handler code for D */
}
```

and

```
void __interrupt(3) C(void)
{
  /* handler code for C */
}
```

Serialization causes higher vector ISRs to be delayed until the completion of lower vector ISRs of the same priority but it gives the following advantage:

- Inhibiting pre-emption between ISRs reduces the worst-case stack and CSA requirements for an application.

3.1.8 Default Interrupt

The default interrupt handler does not require the `__interrupt(<vector>)` or `__interrupt_fast(<vector>)` qualifier and must be declared as a standard C function. The RTA-OSEK Component handles the interrupt context itself in this case.

3.2 Register Settings

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

Register	Required Value
BIV	Interrupt vector table base address
FCX	Free context list. This should be initialized to point to a suitably large linked list of context areas (performed by Tasking C startup code)
PSW.IO	Supervisor (2)

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

Registers Used	Notes
PCXI	Previous context information register
FCX	Free CSA list head pointer
PSW	Processor Status Word
ICR	Interrupt Control Register

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 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): 16

Timing

API max usage (bytes): 16

Extended

API max usage (bytes): 48

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 Stack Mode

The RTA-OSEK Component operates with the PSW.IS bit set to one. This ensures that only one stack is used throughout an application.

3.3.4 Context Save Areas (CSAs)

RTA-OSEK does not currently support calculation of CSA use.

The worst case CSA usage for Task-to-Task switching in standard build is 7 CSAs. In timing and extended builds the figure is 9 CSAs.

For example, consider an application in standard build that has 10 tasks, the code for each of which requires up to 4 CSAs. The combined CSA requirement for those tasks is at most (4 + 7) * 10 = 110 CSAs. This maximum would occur when all but the highest priority task had been started and then pre-empted by the next, higher priority task and when the highest priority task was currently running.

3.3.5 Call Depth Counter

The RTA-OSEK Component can operate with the TriCore call depth counter either enabled or disabled. The RTA-OSEK Component does not alter the call depth counter register settings.

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	**TC1910**
Clock speed (MHz)	40
Code memory	internal scratchpad RAM
Read-only data memory	internal scratchpad RAM
Read-write data memory	external 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	32	32	32	32	32	32
Maximum number of not suspended tasks	32	32	32	32	32	32
Maximum number of priorities	32	32	32	32	32	32
Number of tasks per priority (for BCC2 and ECC2)	n/a	32	32	n/a	32	32
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	32	32	32
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	4294967295					

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	28	28	28	28	28	28
	ROM	204	204	204	204	204	204
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Timing

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
OS overhead	RAM	48	48	48	48	48	48
	ROM	276	276	276	276	276	276
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	66	66	66	66	66	66
	ROM	322	322	322	322	322	322
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

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	36	36	36	36	36	36
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	40	40	40	40	40	40
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	44	52	n/a	44	52
ECC1, Integer task	RAM	n/a	n/a	n/a	28	28	28
	ROM	n/a	n/a	n/a	60	60	60
ECC1, floating-point task	RAM	n/a	n/a	n/a	30	30	30
	ROM	n/a	n/a	n/a	60	60	60
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	30
	ROM	n/a	n/a	n/a	n/a	n/a	68
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	32
	ROM	n/a	n/a	n/a	n/a	n/a	68
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	64	64	64	64	64	64
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	112	112	112	112	112	112
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
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	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	60	60	60	60	60	60
Counter	RAM	4	4	4	4	4	4
	ROM	44	44	44	44	44	44
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20

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	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

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	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	52	52	52	52	52	52
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	56	64	n/a	56	64
ECC1, Integer task	RAM	n/a	n/a	n/a	40	40	40
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating-point task	RAM	n/a	n/a	n/a	42	42	42
	ROM	n/a	n/a	n/a	72	72	72
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	42
	ROM	n/a	n/a	n/a	n/a	n/a	80
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	44
	ROM	n/a	n/a	n/a	n/a	n/a	80

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	132	132	132	132	132	132
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	160	160	160	160	160	160
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
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	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	60	60	60	60	60	60
Counter	RAM	4	4	4	4	4	4
	ROM	44	44	44	44	44	44
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

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	16	16	16	16	16	16
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	64	72	n/a	64	72
ECC1, Integer task	RAM	n/a	n/a	n/a	44	44	44
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	46	46	46
	ROM	n/a	n/a	n/a	80	80	80
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	46
	ROM	n/a	n/a	n/a	n/a	n/a	88
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	48
	ROM	n/a	n/a	n/a	n/a	n/a	88
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	144	144	144	144	144	144
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	172	172	172	172	172	172
Resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	64	64	64	64	64	64
Counter	RAM	4	4	4	4	4	4
	ROM	48	48	48	48	48	48
Message	RAM	11	11	11	31	31	31
	ROM	24	24	24	60	60	60
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28

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	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Arrivalpoint (writable)	RAM	20	20	20	20	20	20
	ROM	20	20	20	20	20	20
Schedule	RAM	20	20	20	20	20	20
	ROM	44	44	44	44	44	44
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

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	128	190	222	134	196	258
	NS		112	168	210	118	174	236
	KL	2	68	124	168	74	130	192

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	26	26	26	26	26	26
ChainTask	SWL	1, 8	134	190	228	140	196	266
	SWH	1, 9	160	218	256	166	224	292
	NSL	8	134	190	228	140	196	266
	NSH	9	148	206	244	154	212	280
Schedule			94	94	122	94	94	122
GetTaskID			42	42	42	42	42	42
GetTaskState			84	84	84	96	96	96
EnableAllInterrupts			34	34	34	34	34	34
DisableAllInterrupts			40	40	40	40	40	40
ResumeAllInterrupts			50	50	50	50	50	50
SuspendAllInterrupts			58	58	58	58	58	58
ResumeOSInterrupts			50	50	50	50	50	50
SuspendOSInterrupts			68	68	68	68	68	68
GetResource	Task	7	30	30	34	30	30	34
	Combined	6	68	68	68	68	68	68
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	74	74	74	74	74	74
	Combined	6	166	166	166	166	166	166
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	128	128	194
	NS		n/a	n/a	n/a	106	106	176
	NS1i	10	n/a	n/a	n/a	64	n/a	n/a
	KL	2	n/a	n/a	n/a	72	72	142
	KL1i	2, 10	n/a	n/a	n/a	30	n/a	n/a
ClearEvent			n/a	n/a	n/a	74	74	74
GetEvent			n/a	n/a	n/a	20	20	20
WaitEvent	<default>		n/a	n/a	n/a	224	224	428
	fp	11	n/a	n/a	n/a	276	276	538
	1i	10	n/a	n/a	n/a	28	n/a	n/a
GetAlarmBase			64	64	64	64	64	64
GetAlarm			104	104	104	104	104	104
SetRelAlarm			132	132	132	132	132	132
SetAbsAlarm			146	146	146	146	146	146
CancelAlarm			98	98	98	98	98	98
InitCounter			66	66	66	66	66	66

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			82	82	82	82	82	82
osek_tick_alarm	<default>		72	72	72	72	72	72
	KL	2	40	40	40	40	40	40
osek_incr_counter			100	100	100	100	100	100
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			238	238	238	238	238	238
ShutdownOS	NoHook	12	42	42	42	42	42	42
	Hook	13	46	46	46	46	46	46
InitCOM			14	14	14	14	14	14
CloseCOM			14	14	14	14	14	14
StartCOM			24	24	24	24	24	24
StopCOM			26	26	26	26	26	26
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	66	66	66	168	168	168
	CCCB	15	168	168	168	168	168	168
GetMessageResource			42	42	42	42	42	42
ReleaseMessageResource			42	42	42	42	42	42
GetMessageStatus			50	50	50	50	50	50
SendMessage	SW CCCA	1, 14	90	90	90	214	214	214
	SW CCCB	1, 15	200	200	200	214	214	214
	NS CCCA	14	90	90	90	214	214	214
	NS CCCB	15	200	200	200	214	214	214
	KL CCCA	2, 14	56	56	56	180	180	180
	KL CCCB	2, 15	166	166	166	180	180	180
main_dispatch	NoHook	12	138	138	176	138	138	176
	Hook	13	174	174	212	174	174	212
sub_dispatch	B1LF	19	32	32	32	32	32	32
	B1HI	20	100	100	100	100	100	100
	B1HF	21	106	106	106	106	106	106
	B2LI	22	n/a	92	118	n/a	92	118
	B2LF	23	n/a	98	124	n/a	98	124
	B2HI	24	n/a	190	282	n/a	190	282
	B2HF	25	n/a	192	288	n/a	192	288
	E1HI	26	n/a	n/a	n/a	386	386	454
	E1HF	27	n/a	n/a	n/a	392	392	460
	E2HI	28	n/a	n/a	n/a	n/a	n/a	454

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	E2HF	29	n/a	n/a	n/a	n/a	n/a	460
ErrorHook support		16	38	38	38	38	38	38
	ServiceID	17	50	50	50	50	50	50
	Parameters	18	80	80	80	80	80	80
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	80	132	174	96	168	204
	NS		58	110	152	74	146	182
	KL	2	24	76	118	40	112	148
ChainTaskset	SWL	1, 8	74	126	166	74	146	180
	SWH	1, 9	96	168	208	96	188	222
	NSL	8	74	126	166	74	146	180
	NSH	9	84	156	196	84	176	210
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			54	54	54	54	54	54
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			64	64	64	64	64	64
TestSubTaskset			56	56	56	56	56	56
TestEquivalentTaskset			54	54	54	54	54	54
TickSchedule	SW	1	132	136	136	136	136	136
	NS		112	116	116	116	116	116
	KL	2	78	82	82	82	82	82
AdvanceSchedule	SW	1	118	116	116	116	116	116
	NS		98	96	96	96	96	96
	KL	2	64	62	62	62	62	62
StartSchedule			76	76	76	76	76	76
StopSchedule			64	64	64	64	64	64
GetScheduleStatus			92	92	92	92	92	92
GetScheduleValue			72	72	72	72	72	72
GetScheduleNext			20	20	20	20	20	20
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			16	16	16	16	16	16
GetArrivalpointTasksetRef			14	14	14	14	14	14
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			16	16	16	16	16	16

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestArrivalpointWritable			44	44	44	44	44	44
GetExecutionTime			14	14	14	14	14	14
GetLargestExecutionTime			16	16	16	16	16	16
ResetLargestExecutionTime			14	14	14	14	14	14
GetStackOffset			26	26	26	26	26	26
Floating point support			68	68	68	68	68	68
Interrupt support			66	66	66	66	66	66
Context save			50	50	50	50	50	50
Context restore			162	162	162	162	162	162

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	128	190	222	134	196	258
	NS		112	168	210	118	174	236
	KL	2	68	124	168	74	130	192
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	26	26	26	26	26	26
ChainTask	SWL	1, 8	134	190	228	140	196	266
	SWH	1, 9	160	218	256	166	224	292
	NSL	8	134	190	228	140	196	266
	NSH	9	148	206	244	154	212	280
Schedule			116	116	144	116	116	144
GetTaskID			42	42	42	42	42	42
GetTaskState			84	84	84	96	96	96
EnableAllInterrupts			34	34	34	34	34	34
DisableAllInterrupts			40	40	40	40	40	40
ResumeAllInterrupts			50	50	50	50	50	50
SuspendAllInterrupts			58	58	58	58	58	58
ResumeOSInterrupts			50	50	50	50	50	50
SuspendOSInterrupts			68	68	68	68	68	68
GetResource	Task	7	30	30	34	30	30	34

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Combined	6	68	68	68	68	68	68
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	98	98	98	98	98	98
	Combined	6	212	212	212	212	212	212
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	128	128	194
	NS		n/a	n/a	n/a	106	106	176
	NS1i	10	n/a	n/a	n/a	64	n/a	n/a
	KL	2	n/a	n/a	n/a	72	72	142
	KL1i	2, 10	n/a	n/a	n/a	30	n/a	n/a
ClearEvent			n/a	n/a	n/a	74	74	74
GetEvent			n/a	n/a	n/a	20	20	20
WaitEvent	<default>		n/a	n/a	n/a	320	320	522
	fp	11	n/a	n/a	n/a	346	346	576
	1i	10	n/a	n/a	n/a	112	n/a	n/a
GetAlarmBase			64	64	64	64	64	64
GetAlarm			104	104	104	104	104	104
SetRelAlarm			132	132	132	132	132	132
SetAbsAlarm			146	146	146	146	146	146
CancelAlarm			98	98	98	98	98	98
InitCounter			66	66	66	66	66	66
GetCounterValue			82	82	82	82	82	82
osek_tick_alarm	<default>		72	72	72	72	72	72
	KL	2	40	40	40	40	40	40
osek_incr_counter			100	100	100	100	100	100
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			292	292	292	292	292	292
ShutdownOS	NoHook	12	42	42	42	42	42	42
	Hook	13	46	46	46	46	46	46
InitCOM			14	14	14	14	14	14
CloseCOM			14	14	14	14	14	14
StartCOM			24	24	24	24	24	24
StopCOM			26	26	26	26	26	26
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	66	66	66	168	168	168
	CCCB	15	168	168	168	168	168	168

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageResource			42	42	42	42	42	42
ReleaseMessageResource			42	42	42	42	42	42
GetMessageStatus			50	50	50	50	50	50
SendMessage	SW CCCA	1, 14	90	90	90	214	214	214
	SW CCCB	1, 15	200	200	200	214	214	214
	NS CCCA	14	90	90	90	214	214	214
	NS CCCB	15	200	200	200	214	214	214
	KL CCCA	2, 14	56	56	56	180	180	180
	KL CCCB	2, 15	166	166	166	180	180	180
main_dispatch	NoHook	12	188	188	228	188	188	228
	Hook	13	232	232	272	232	232	272
sub_dispatch	B1LF	19	20	20	20	20	20	20
	B1HI	20	94	94	94	94	94	94
	B1HF	21	102	102	102	102	102	102
	B2LI	22	n/a	60	94	n/a	60	94
	B2LF	23	n/a	66	100	n/a	66	100
	B2HI	24	n/a	136	222	n/a	136	222
	B2HF	25	n/a	138	228	n/a	138	228
	E1HI	26	n/a	n/a	n/a	384	384	452
	E1HF	27	n/a	n/a	n/a	390	390	458
	E2HI	28	n/a	n/a	n/a	n/a	n/a	452
	E2HF	29	n/a	n/a	n/a	n/a	n/a	458
ErrorHook support		16	38	38	38	38	38	38
	ServiceID	17	50	50	50	50	50	50
	Parameters	18	80	80	80	80	80	80
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	80	80	80	80	80	80
Timing_termination		4	78	78	78	78	78	78
ActivateTaskset	SW	1	80	132	174	96	168	204
	NS		58	110	152	74	146	182
	KL	2	24	76	118	40	112	148
ChainTaskset	SWL	1, 8	74	126	166	74	146	180
	SWH	1, 9	96	168	208	96	188	222
	NSL	8	74	126	166	74	146	180
	NSH	9	84	156	196	84	176	210
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			54	54	54	54	54	54

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			64	64	64	64	64	64
TestSubTaskset			56	56	56	56	56	56
TestEquivalentTaskset			54	54	54	54	54	54
TickSchedule	SW	1	132	136	136	136	136	136
	NS		112	116	116	116	116	116
	KL	2	78	82	82	82	82	82
AdvanceSchedule	SW	1	118	116	116	116	116	116
	NS		98	96	96	96	96	96
	KL	2	64	62	62	62	62	62
StartSchedule			76	76	76	76	76	76
StopSchedule			64	64	64	64	64	64
GetScheduleStatus			92	92	92	92	92	92
GetScheduleValue			72	72	72	72	72	72
GetScheduleNext			20	20	20	20	20	20
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			16	16	16	16	16	16
GetArrivalpointTasksetRef			14	14	14	14	14	14
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			16	16	16	16	16	16
TestArrivalpointWritable			44	44	44	44	44	44
GetExecutionTime			110	110	110	110	110	110
GetLargestExecutionTime			22	22	22	22	22	22
ResetLargestExecutionTime			20	20	20	20	20	20
GetStackOffset			26	26	26	26	26	26
Floating point support			68	68	68	68	68	68
Interrupt support			180	180	180	180	180	180
Context save			50	50	50	50	50	50
Context restore			162	162	162	162	162	162

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	242	306	348	256	312	374
	NS		276	338	380	290	344	406
	KL	2	156	218	260	170	224	286
TerminateTask	LExt	3	120	120	120	120	120	120
	H	5	144	144	144	144	144	144
ChainTask	SWL	1, 8	276	342	380	292	348	418
	SWH	1, 9	308	378	416	324	384	450
	NSL	8	314	380	418	330	386	456
	NSH	9	340	410	448	356	416	482
Schedule			252	252	284	252	252	284
GetTaskID			58	58	58	58	58	58
GetTaskState			240	240	240	244	244	244
EnableAllInterrupts			50	50	50	50	50	50
DisableAllInterrupts			56	56	56	56	56	56
ResumeAllInterrupts			102	102	102	102	102	102
SuspendAllInterrupts			74	74	74	74	74	74
ResumeOSInterrupts			102	102	102	102	102	102
SuspendOSInterrupts			84	84	84	84	84	84
GetResource	Task	7	358	358	326	358	358	326
	Combined	6	336	336	336	336	336	336
	CLEx	3	302	302	302	302	302	302
ReleaseResource	Task	7	334	334	334	334	334	334
	Combined	6	428	428	428	428	428	428
	CLEx	3	294	294	294	294	294	294
SetEvent	SW	1	n/a	n/a	n/a	294	294	370
	NS		n/a	n/a	n/a	330	330	406
	NS1i	10	n/a	n/a	n/a	230	n/a	n/a
	KL	2	n/a	n/a	n/a	212	212	294
	KL1i	2, 10	n/a	n/a	n/a	162	n/a	n/a
ClearEvent			n/a	n/a	n/a	156	156	156
GetEvent			n/a	n/a	n/a	160	160	160
WaitEvent	<default>		n/a	n/a	n/a	448	448	654
	fp	11	n/a	n/a	n/a	476	476	702

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	238	n/a	n/a
GetAlarmBase			196	196	196	196	196	196
GetAlarm			182	182	182	182	182	182
SetRelAlarm			238	238	238	238	238	238
SetAbsAlarm			252	252	252	252	252	252
CancelAlarm			170	170	170	170	170	170
InitCounter			234	234	234	234	234	234
GetCounterValue			214	214	214	214	214	214
osek_tick_alarm	<default>		112	112	112	112	112	112
	KL	2	40	40	40	40	40	40
osek_incr_counter			100	100	100	100	100	100
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			308	308	308	308	308	308
ShutdownOS	NoHook	12	54	54	54	54	54	54
	Hook	13	58	58	58	58	58	58
InitCOM			14	14	14	14	14	14
CloseCOM			14	14	14	14	14	14
StartCOM			42	42	42	42	42	42
StopCOM			56	56	56	56	56	56
ReadFlag			40	40	40	40	40	40
ResetFlag			44	44	44	44	44	44
ReceiveMessage	CCCA	14	164	164	164	264	264	264
	CCCB	15	264	264	264	264	264	264
GetMessageResource			84	84	84	84	84	84
ReleaseMessageResource			84	84	84	84	84	84
GetMessageStatus			92	92	92	92	92	92
SendMessage	SW CCCA	1, 14	202	202	202	330	330	330
	SW CCCB	1, 15	316	316	316	330	330	330
	NS CCCA	14	202	202	202	330	330	330
	NS CCCB	15	316	316	316	330	330	330
	KL CCCA	2, 14	134	134	134	248	248	248
	KL CCCB	2, 15	238	238	238	248	248	248
main_dispatch	NoHook	12	188	188	228	188	188	228
	Hook	13	232	232	272	232	232	272
sub_dispatch	B1LF	19	20	20	20	20	20	20
	B1HI	20	94	94	94	94	94	94
	B1HF	21	102	102	102	102	102	102

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	B2LI	22	n/a	60	94	n/a	60	94
	B2LF	23	n/a	66	100	n/a	66	100
	B2HI	24	n/a	136	222	n/a	136	222
	B2HF	25	n/a	138	228	n/a	138	228
	E1HI	26	n/a	n/a	n/a	384	384	452
	E1HF	27	n/a	n/a	n/a	390	390	458
	E2HI	28	n/a	n/a	n/a	n/a	n/a	452
	E2HF	29	n/a	n/a	n/a	n/a	n/a	458
ErrorHook support		16	110	110	110	110	110	110
	ServiceID	17	126	126	126	126	126	126
	Parameters	18	154	154	154	154	154	154
validity_checks		3	30	30	30	30	30	30
Timing_dispatch		4	80	80	80	80	80	80
Timing_termination		4	78	78	78	78	78	78
ActivateTaskset	SW	1	326	408	440	372	440	496
	NS		362	442	478	410	480	534
	KL	2	242	330	364	294	372	432
ChainTaskset	SWL	1, 8	380	468	514	418	494	536
	SWH	1, 9	418	524	560	456	550	592
	NSL	8	438	520	564	464	542	588
	NSH	9	464	562	596	490	584	632
GetTasksetRef			130	130	130	130	130	130
MergeTaskset			306	306	306	306	306	306
AssignTaskset			214	214	214	214	214	214
RemoveTaskset			316	316	316	316	316	316
TestSubTaskset			318	318	318	318	318	318
TestEquivalentTaskset			316	316	316	316	316	316
TickSchedule	SW	1	352	274	274	274	274	274
	NS		386	342	342	342	342	342
	KL	2	284	202	202	202	202	202
AdvanceSchedule	SW	1	352	276	276	276	276	276
	NS		386	338	338	338	338	338
	KL	2	284	206	206	206	206	206
StartSchedule			254	254	254	254	254	254
StopSchedule			210	210	210	210	210	210
GetScheduleStatus			244	244	244	244	244	244
GetScheduleValue			204	204	204	204	204	204

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleNext			108	108	108	108	108	108
SetScheduleNext			200	200	200	200	200	200
GetArrivalpointDelay			150	150	150	150	150	150
SetArrivalpointDelay			166	166	166	166	166	166
GetArrivalpointTasksetRef			148	148	148	148	148	148
GetArrivalpointNext			150	150	150	150	150	150
SetArrivalpointNext			222	222	222	222	222	222
TestArrivalpointWritable			166	166	166	166	166	166
GetExecutionTime			166	166	166	166	166	166
GetLargestExecutionTime			112	112	112	112	112	112
ResetLargestExecutionTime			106	106	106	106	106	106
GetStackOffset			26	26	26	26	26	26
Floating point support			68	68	68	68	68	68
Interrupt support			180	180	180	180	180	180
Context save			50	50	50	50	50	50
Context restore			162	162	162	162	162	162

Notes

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

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

4.2.4 Reserved Hardware Resources

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

4.3 Performance

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	99	183	269	101	128	233
	NS	97	180	265	99	124	228
	KL	87	170	257	130	139	254
TerminateTask	LExt	0	0	0	0	0	0
	H	231	254	270	233	233	234
ChainTask	SWL	439	471	698	552	539	782
	SWH	556	637	845	670	695	863
	NSL	402	469	684	485	522	759
	NSH	555	608	836	634	653	884
Schedule	SW	133	133	149	133	133	149
GetTaskID		97	97	97	97	97	108
GetTaskState		137	137	137	192	171	154
EnableAllInterrupts		66	66	66	68	68	68
DisableAllInterrupts		38	38	38	48	48	48
ResumeAllInterrupts		87	84	87	87	87	87
SuspendAllInterrupts		93	93	93	92	92	99
ResumeOSInterrupts		87	87	87	88	88	99
SuspendOSInterrupts		95	95	95	77	77	77
GetResource	Task	79	102	80	79	79	80
	Combined	86	86	86	90	90	90
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	106	105	106	110	110	110
	Combined	139	147	139	134	135	134
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	186	169	171
	NS	n/a	n/a	n/a	203	176	161
	KL	n/a	n/a	n/a	139	139	181
ClearEvent		n/a	n/a	n/a	109	109	109
GetEvent		n/a	n/a	n/a	90	90	90
WaitEvent	<default>	n/a	n/a	n/a	655	663	728
	fp	n/a	n/a	n/a	669	677	795
GetAlarmBase		139	136	136	115	115	134
GetAlarm		163	163	163	193	152	152

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		147	147	149	156	156	158
SetAbsAlarm		144	144	144	133	133	134
CancelAlarm		117	105	99	105	105	105
InitCounter		105	105	105	109	109	109
GetCounterValue		122	122	122	125	125	125
osek_tick_alarm	<default>	172	172	172	172	173	173
	KL	153	153	153	163	207	207
osek_incr_counter		54	54	54	104	59	59
GetActiveApplicationMode		38	38	38	38	38	38
StartOS		1325	1325	1371	1325	1337	1368
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	54	54	54	54	54	58
InitCOM		38	38	38	31	31	31
CloseCOM		38	38	38	31	31	31
StartCOM		73	73	73	207	200	201
StopCOM		65	65	65	55	55	56
ReadFlag		n/a	n/a	n/a	49	49	49
ResetFlag		n/a	n/a	n/a	41	38	38
ReceiveMessage		92	92	92	674	686	688
GetMessageResource		n/a	n/a	n/a	185	201	184
ReleaseMessageResource		n/a	n/a	n/a	244	211	212
GetMessageStatus		n/a	n/a	n/a	153	137	115
SendMessage	SW	169	254	340	723	748	831
	NS	185	271	374	720	748	852
	KL	139	222	341	691	676	820
ActivateTaskset	SW	92	329	415	98	315	424
	NS	81	349	404	87	303	454
	KL	65	304	391	73	289	430
	SW2	92	329	455	98	325	424
	NS2	81	349	404	87	303	444
	KL2	65	304	389	73	289	430
ChainTaskset	SWL	409	615	853	490	717	925
	SWH	494	744	987	626	873	1103
	NSL	424	610	861	542	783	975
	NSH	491	766	1000	634	840	1052
GetTasksetRef		92	91	91	91	91	91

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		115	115	129	119	119	119
AssignTaskset		84	122	84	88	88	88
RemoveTaskset		116	116	116	120	120	162
TestSubTaskset		117	117	117	155	121	121
TestEquivalentTaskset		107	107	106	111	111	111
TickSchedule	SW	209	476	578	243	496	632
	NS	192	474	590	244	500	635
	KL	175	461	564	260	510	620
	SW2	223	500	580	242	460	609
	NS2	192	475	582	276	462	632
	KL2	175	459	564	228	484	587
AdvanceSchedule	SW	168	438	523	213	469	608
	NS	153	423	510	234	456	590
	KL	148	416	538	196	450	586
	SW2	168	440	529	243	431	575
	NS2	179	425	508	200	416	561
	KL2	148	449	538	196	412	555
StartSchedule		147	134	143	135	135	135
StopSchedule		126	127	126	129	129	129
GetScheduleStatus		193	162	196	171	170	170
GetScheduleValue		117	117	117	120	120	120
GetScheduleNext		82	82	82	86	86	86
SetScheduleNext		81	81	81	82	109	81
GetArrivalpointDelay		67	67	71	71	71	71
SetArrivalpointDelay		69	69	69	69	69	69
GetArrivalpointTasksetRef		52	52	52	52	52	52
GetArrivalpointNext		67	67	71	71	71	93
SetArrivalpointNext		69	69	69	69	69	69
TestArrivalpointWritable		62	62	62	62	62	62
GetExecutionTime		41	41	41	41	41	67
GetLargestExecutionTime		61	62	61	61	61	61
ResetLargestExecutionTime		63	63	63	63	63	63
GetStackOffset		46	46	46	46	46	46

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	99	183	268	101	129	231
	NS	97	198	264	99	124	228
	KL	87	170	255	118	114	222
TerminateTask	LExt	0	0	0	0	0	0
	H	608	594	590	619	605	574
ChainTask	SWL	790	910	1162	953	970	1201
	SWH	935	1046	1290	1067	1067	1361
	NSL	808	883	1099	902	938	1160
	NSH	917	1087	1276	1110	1114	1376
Schedule	SW	133	168	148	133	133	148
GetTaskID		97	135	97	97	97	97
GetTaskState		154	136	136	153	154	154
EnableAllInterrupts		100	66	108	68	94	94
DisableAllInterrupts		38	39	38	48	48	48
ResumeAllInterrupts		84	84	84	87	87	87
SuspendAllInterrupts		93	93	93	122	92	92
ResumeOSInterrupts		87	87	87	122	88	88
SuspendOSInterrupts		125	95	125	77	92	88
GetResource	Task	79	79	80	79	79	80
	Combined	86	86	86	89	90	90
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	105	104	105	109	109	109
	Combined	139	141	140	134	134	134
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	169	169	171
	NS	n/a	n/a	n/a	160	158	159
	KL	n/a	n/a	n/a	139	139	139
ClearEvent		n/a	n/a	n/a	109	109	109
GetEvent		n/a	n/a	n/a	90	115	90
WaitEvent	<default>	n/a	n/a	n/a	986	973	1094
	fp	n/a	n/a	n/a	1016	1027	1133
GetAlarmBase		138	141	139	126	126	123
GetAlarm		185	163	163	152	152	152

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		149	149	147	156	156	156
SetAbsAlarm		160	183	144	133	133	134
CancelAlarm		99	99	99	105	105	105
InitCounter		105	142	105	109	109	109
GetCounterValue		122	122	122	125	153	125
osek_tick_alarm	<default>	172	172	172	190	185	173
	KL	153	153	153	163	163	163
osek_incr_counter		54	54	54	59	59	59
GetActiveApplicationMode		38	38	38	38	38	38
StartOS		3331	3327	3371	3371	3327	3332
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	54	54	54	54	54	54
InitCOM		38	38	38	31	31	31
CloseCOM		38	38	38	31	31	31
StartCOM		73	73	73	200	200	200
StopCOM		65	86	65	55	55	55
ReadFlag		n/a	n/a	n/a	49	49	49
ResetFlag		n/a	n/a	n/a	38	38	38
ReceiveMessage		92	92	92	686	652	646
GetMessageResource		n/a	n/a	n/a	201	208	185
ReleaseMessageResource		n/a	n/a	n/a	208	209	238
GetMessageStatus		n/a	n/a	n/a	115	115	115
SendMessage	SW	169	278	338	727	752	855
	NS	185	271	354	694	743	838
	KL	140	240	307	689	673	780
ActivateTaskset	SW	96	337	423	97	313	429
	NS	85	357	412	87	303	413
	KL	69	312	397	72	288	399
	SW2	96	337	429	97	313	421
	NS2	85	357	412	87	303	413
	KL2	69	312	397	102	288	397
ChainTaskset	SWL	817	1059	1271	946	1108	1400
	SWH	935	1176	1411	1112	1259	1544
	NSL	816	1018	1272	989	1159	1407
	NSH	933	1186	1410	1066	1285	1502
GetTasksetRef		91	91	119	91	91	92

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		129	119	119	119	119	119
AssignTaskset		88	88	88	118	88	88
RemoveTaskset		120	162	120	120	120	120
TestSubTaskset		121	121	121	121	121	121
TestEquivalentTaskset		105	111	111	111	111	111
TickSchedule	SW	193	528	606	241	498	646
	NS	195	501	570	243	499	647
	KL	179	469	554	227	483	615
	SW2	193	528	612	241	460	611
	NS2	195	519	572	283	459	612
	KL2	185	509	556	227	453	590
AdvanceSchedule	SW	178	456	542	237	498	575
	NS	165	443	554	199	455	559
	KL	160	439	550	195	451	591
	SW2	180	458	542	215	430	545
	NS2	165	483	554	199	417	530
	KL2	200	441	522	221	411	526
StartSchedule		137	137	173	135	146	135
StopSchedule		126	126	126	129	130	129
GetScheduleStatus		170	168	201	171	204	170
GetScheduleValue		117	117	117	120	120	120
GetScheduleNext		86	86	86	86	86	86
SetScheduleNext		81	81	81	81	81	105
GetArrivalpointDelay		71	71	71	71	71	71
SetArrivalpointDelay		69	69	69	69	69	69
GetArrivalpointTasksetRef		52	52	52	52	52	52
GetArrivalpointNext		93	71	71	91	71	71
SetArrivalpointNext		69	69	69	69	69	69
TestArrivalpointWritable		62	62	62	62	62	62
GetExecutionTime		211	245	211	213	211	211
GetLargestExecutionTime		123	97	97	97	97	97
ResetLargestExecutionTime		82	82	82	82	82	110
GetStackOffset		46	46	46	46	46	46

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	370	451	534	375	394	487
	NS	460	565	660	462	480	576
	KL	370	433	560	359	375	476
TerminateTask	LExt	582	608	592	573	573	607
	H	753	745	745	735	735	760
ChainTask	SWL	1160	1324	1504	1306	1322	1560
	SWH	1347	1418	1623	1441	1474	1729
	NSL	1271	1343	1535	1351	1405	1637
	NSH	1391	1480	1699	1507	1580	1841
Schedule	SW	203	236	217	202	202	220
GetTaskID		114	144	114	114	152	114
GetTaskState		427	430	449	438	437	437
EnableAllInterrupts		79	79	79	81	81	81
DisableAllInterrupts		49	50	50	70	60	60
ResumeAllInterrupts		145	116	116	153	119	119
SuspendAllInterrupts		110	111	110	109	109	109
ResumeOSInterrupts		119	119	119	120	120	120
SuspendOSInterrupts		112	113	112	94	94	94
GetResource	Task	806	752	450	845	811	509
	Combined	367	365	383	439	437	455
	CLEx	441	455	443	514	517	555
ReleaseResource	Task	438	401	402	496	457	457
	Combined	372	372	387	453	483	457
	CLEx	374	411	376	434	433	434
SetEvent	SW	n/a	n/a	n/a	448	449	469
	NS	n/a	n/a	n/a	471	471	471
	KL	n/a	n/a	n/a	421	421	426
ClearEvent		n/a	n/a	n/a	159	160	159
GetEvent		n/a	n/a	n/a	410	416	378
WaitEvent	<default>	n/a	n/a	n/a	1278	1313	1391
	fp	n/a	n/a	n/a	1283	1344	1387
GetAlarmBase		314	302	313	332	363	321
GetAlarm		348	346	348	352	355	388

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
SetRelAlarm		381	381	380	361	398	363
SetAbsAlarm		332	333	371	341	338	339
CancelAlarm		294	313	291	290	289	290
InitCounter		507	504	504	534	528	542
GetCounterValue		282	281	284	291	289	319
osek_tick_alarm	<default>	188	188	188	196	194	194
	KL	155	160	160	191	153	153
osek_incr_counter		67	67	69	54	54	54
GetActiveApplicationMode		38	38	38	71	38	38
StartOS		3447	3445	3487	3443	3452	3487
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	60	60	60	60	60	60
InitCOM		40	38	38	38	38	38
CloseCOM		38	38	38	38	38	38
StartCOM		109	86	86	254	236	236
StopCOM		92	130	92	98	98	98
ReadFlag		n/a	n/a	n/a	103	105	103
ResetFlag		n/a	n/a	n/a	78	78	78
ReceiveMessage		255	256	254	804	807	776
GetMessageResource		n/a	n/a	n/a	662	704	684
ReleaseMessageResource		n/a	n/a	n/a	632	668	660
GetMessageStatus		n/a	n/a	n/a	198	198	198
SendMessage	SW	614	724	809	1152	1158	1265
	NS	727	822	866	1239	1263	1325
	KL	552	642	721	1057	1074	1175
ActivateTaskset	SW	426	672	729	437	623	747
	NS	492	705	807	502	654	768
	KL	396	592	738	402	534	708
	SW2	432	642	746	466	623	707
	NS2	526	704	791	528	660	768
	KL2	396	594	758	402	534	708
ChainTaskset	SWL	1268	1481	1706	1426	1572	1837
	SWH	1372	1673	1845	1527	1629	1967
	NSL	1331	1579	1777	1447	1576	1930
	NSH	1466	1625	1922	1560	1713	1997
GetTasksetRef		365	352	353	353	352	352

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		190	191	191	191	190	209
AssignTaskset		140	139	140	174	139	139
RemoveTaskset		225	193	191	191	191	193
TestSubTaskset		187	227	189	189	188	187
TestEquivalentTaskset		177	219	179	179	198	177
TickSchedule	SW	361	807	973	623	823	993
	NS	491	955	1097	790	970	1147
	KL	305	809	943	608	807	979
	SW2	343	840	993	623	755	929
	NS2	483	990	1113	782	898	1074
	KL2	293	867	991	608	739	923
AdvanceSchedule	SW	362	878	1011	658	857	1029
	NS	483	939	1082	753	952	1158
	KL	369	815	973	630	829	1001
	SW2	355	843	987	658	789	965
	NS2	453	939	1106	753	884	1060
	KL2	357	879	991	654	761	937
StartSchedule		228	225	212	210	209	209
StopSchedule		201	198	165	167	166	166
GetScheduleStatus		213	213	213	214	213	255
GetScheduleValue		163	164	163	171	167	169
GetScheduleNext		108	109	108	108	108	108
SetScheduleNext		148	149	149	149	148	148
GetArrivalpointDelay		100	100	100	100	130	135
SetArrivalpointDelay		135	136	135	166	135	135
GetArrivalpointTasksetRef		81	81	81	81	82	81
GetArrivalpointNext		95	95	113	95	95	95
SetArrivalpointNext		158	159	158	152	185	188
TestArrivalpointWritable		82	83	89	88	88	88
GetExecutionTime		237	238	237	231	259	279
GetLargestExecutionTime		336	336	336	352	334	334
ResetLargestExecutionTime		320	320	321	324	320	320
GetStackOffset		46	46	46	45	46	46

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	93	93	94	95	93	95
	Cat 2	124	126	126	124	124	124

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	93	93	93	95	93	93
	Cat 2	399	388	388	384	384	384

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	93	93	95	95	95	93
	Cat 2	388	384	384	384	384	384

4.3.4 Task Switching Times

Task switching time is the time between the last instruction of the previous task and the first instruction of the next task. The switching time differs, depending on the switching contexts (e.g. an `ActivateTask()` versus a `ChainTask()`).

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

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

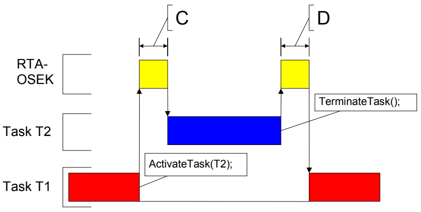

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

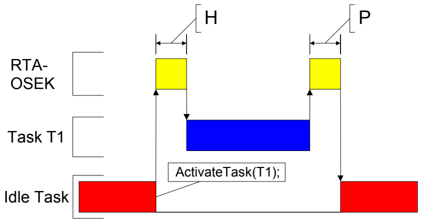

Figure 3: Task Activation from Idle Task

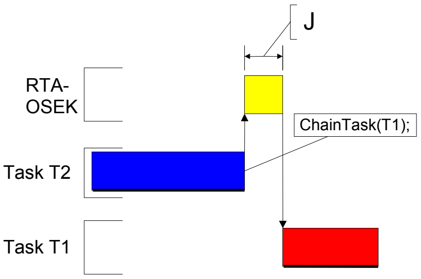

Figure 2: Task Chaining

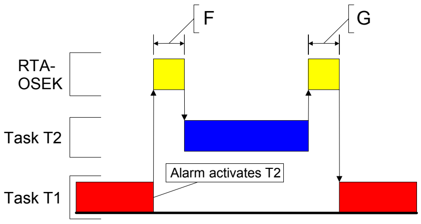

Figure 4: Task Activation from an Alarm

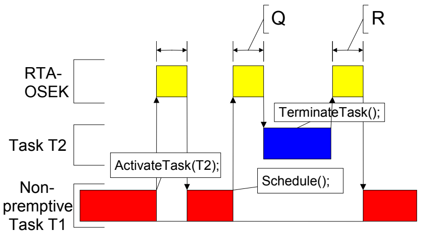

Figure 5: Non-Premptive Task Calls Schedule()

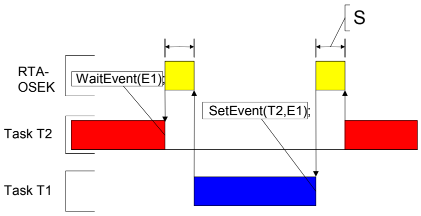

Figure 7: Waiting Task Activated by SetEvent()

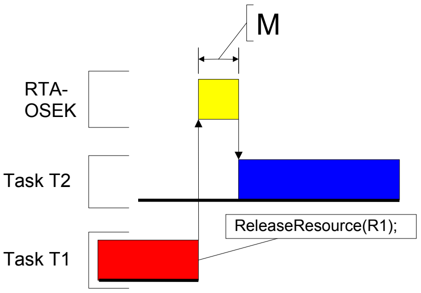

Figure 6: Blocked Task Activated by ReleaseResource()

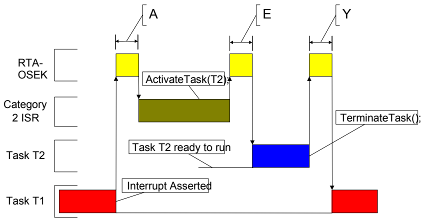

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	79	139	211	83	140	228
Figure 1: D	Heavy, Basic/Extended	256	281	372	334	294	370
ChainTask	Light, Basic	229	347	548	242	358	584
Figure 2: J	Heavy, Basic/Extended	707	904	1218	750	870	1183
Pre-emption	Light, Basic	263	387	560	233	381	613
Figure 1: C	Heavy, Basic/Extended	420	518	704	561	557	763
From idle task	Light, Basic	233	353	602	233	351	589
Figure 3: H	Heavy, Basic/Extended	414	488	676	487	557	808
Triggered by alarm	Light, Basic	385	481	688	401	515	712
Figure 4: F	Heavy, Basic/Extended	515	607	832	635	701	920
Schedule	Light, Basic	214	250	390	212	248	404
Figure 5: Q	Heavy, Basic/Extended	394	360	516	470	478	608
Release resource	Light, Basic	249	285	447	275	281	408
Figure 6: M	Heavy, Basic/Extended	401	393	540	533	511	625
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	693	691	937

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	225	261	384	219	255	375
Figure 8: E	Heavy, Basic/Extended	377	371	494	477	485	599

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	439	477	535	434	501	529
Figure 1: D	Heavy, Basic/Extended	586	654	724	653	673	726
ChainTask	Light, Basic	673	766	1021	695	783	1025
Figure 2: J	Heavy, Basic/Extended	1493	1657	1963	1513	1651	2012
Pre-emption	Light, Basic	577	663	877	578	650	935
Figure 1: C	Heavy, Basic/Extended	696	801	1046	852	844	1123
From idle task	Light, Basic	578	691	915	598	651	923
Figure 3: H	Heavy, Basic/Extended	710	816	999	861	850	1113
Triggered by alarm	Light, Basic	700	816	1036	696	814	1062
Figure 4: F	Heavy, Basic/Extended	817	936	1197	1026	1020	1301
Schedule	Light, Basic	541	597	742	557	587	741
Figure 5: Q	Heavy, Basic/Extended	668	687	857	813	815	987
Release resource	Light, Basic	579	616	717	609	581	716
Figure 6: M	Heavy, Basic/Extended	706	738	876	829	832	1001
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	925	973	1192
From category 2 ISR	Light, Basic	833	842	967	823	832	968
Figure 8: E	Heavy, Basic/Extended	964	944	1078	1071	1079	1255

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	589	675	726	607	650	722
Figure 1: D	Heavy, Basic/Extended	759	812	888	824	823	910
ChainTask	Light, Basic	1037	1133	1365	1035	1150	1391
Figure 2: J	Heavy, Basic/Extended	2073	2215	2490	2062	2154	2470
Pre-emption	Light, Basic	844	933	1157	854	948	1168
Figure 1: C	Heavy, Basic/Extended	979	1065	1318	1143	1092	1360
From idle task	Light, Basic	842	907	1149	864	910	1148
Figure 3: H	Heavy, Basic/Extended	985	1058	1300	1114	1150	1335
Triggered by alarm	Light, Basic	958	1051	1272	982	1076	1302
Figure 4: F	Heavy, Basic/Extended	1162	1226	1439	1279	1270	1526
Schedule	Light, Basic	627	616	794	625	651	808
Figure 5: Q	Heavy, Basic/Extended	750	746	935	902	878	1006
Release resource	Light, Basic	851	833	1003	929	937	1082
Figure 6: M	Heavy, Basic/Extended	1008	1018	1143	1184	1160	1320
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1356	1383	1628
From category 2 ISR	Light, Basic	864	859	987	852	858	1020
Figure 8: E	Heavy, Basic/Extended	1008	1047	1170	1121	1115	1264

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 Planner 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		12	12	13	12	12	13
BCC1 lightweight, floating-point		13	13	14	13	13	14
BCC1 heavyweight, integer		28	28	29	28	28	29
BCC1 heavyweight, floating-point		28	28	29	28	28	29
BCC2 lightweight, integer		n/a	13	14	n/a	13	14
BCC2 lightweight, floating-point		n/a	13	14	n/a	13	14
BCC2 heavyweight, integer		n/a	28	29	n/a	28	29
BCC2 heavyweight, floating-point		n/a	28	29	n/a	28	29
ECC1 heavyweight, integer		n/a	n/a	n/a	29	29	30
ECC1 heavyweight, floating-point		n/a	n/a	n/a	29	29	30
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	30
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	30
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		13	13	13	13	13	13
BCC1 lightweight, floating-point		14	14	14	14	14	14
BCC1 heavyweight, integer		29	29	29	29	29	29
BCC1 heavyweight, floating-point		29	29	29	29	29	29
BCC2 lightweight, integer		n/a	14	14	n/a	14	14
BCC2 lightweight, floating-point		n/a	14	14	n/a	14	14
BCC2 heavyweight, integer		n/a	29	29	n/a	29	29
BCC2 heavyweight, floating-point		n/a	29	29	n/a	29	29
ECC1 heavyweight, integer		n/a	n/a	n/a	30	30	30
ECC1 heavyweight, floating-point		n/a	n/a	n/a	30	30	30
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	30
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	30

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	22	21	21	22
BCC1 lightweight, floating-point		22	22	23	22	22	23
BCC1 heavyweight, integer		37	37	38	37	37	38
BCC1 heavyweight, floating-point		37	37	38	37	37	38
BCC2 lightweight, integer		n/a	22	23	n/a	22	23
BCC2 lightweight, floating-point		n/a	22	23	n/a	22	23
BCC2 heavyweight, integer		n/a	37	38	n/a	37	38
BCC2 heavyweight, floating-point		n/a	37	38	n/a	37	38
ECC1 heavyweight, integer		n/a	n/a	n/a	38	38	39
ECC1 heavyweight, floating-point		n/a	n/a	n/a	38	38	39
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	39
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	39
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		30	30	30	30	30	30
BCC1 lightweight, floating-point		31	31	31	31	31	31
BCC1 heavyweight, integer		46	46	46	46	46	46
BCC1 heavyweight, floating-point		46	46	46	46	46	46
BCC2 lightweight, integer		n/a	31	31	n/a	31	31
BCC2 lightweight, floating-point		n/a	31	31	n/a	31	31
BCC2 heavyweight, integer		n/a	46	46	n/a	46	46
BCC2 heavyweight, floating-point		n/a	46	46	n/a	46	46
ECC1 heavyweight, integer		n/a	n/a	n/a	47	47	47
ECC1 heavyweight, floating-point		n/a	n/a	n/a	47	47	47
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	47
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	47

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		21	21	22	21	21	22
BCC1 lightweight, floating-point		22	22	23	22	22	23
BCC1 heavyweight, integer		37	37	38	37	37	38
BCC1 heavyweight, floating-point		37	37	38	37	37	38
BCC2 lightweight, integer		n/a	22	23	n/a	22	23
BCC2 lightweight, floating-point		n/a	22	23	n/a	22	23
BCC2 heavyweight, integer		n/a	37	38	n/a	37	38
BCC2 heavyweight, floating-point		n/a	37	38	n/a	37	38
ECC1 heavyweight, integer		n/a	n/a	n/a	54	54	55
ECC1 heavyweight, floating-point		n/a	n/a	n/a	54	54	55
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	55
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	55
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		30	30	30	30	30	30
BCC1 lightweight, floating-point		31	31	31	31	31	31
BCC1 heavyweight, integer		46	46	46	46	46	46
BCC1 heavyweight, floating-point		46	46	46	46	46	46
BCC2 lightweight, integer		n/a	31	31	n/a	31	31
BCC2 lightweight, floating-point		n/a	31	31	n/a	31	31
BCC2 heavyweight, integer		n/a	46	46	n/a	46	46
BCC2 heavyweight, floating-point		n/a	46	46	n/a	46	46
ECC1 heavyweight, integer		n/a	n/a	n/a	63	63	63
ECC1 heavyweight, floating-point		n/a	n/a	n/a	63	63	63
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	63
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	63

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.