RTA-OSEK

Binding Manual: TriCore/HighTec

Contact Details

ETAS Group www.etasgroup.com	**ETAS GmbH** 70469 Stuttgart, Germany Tel.:+49 711 89661-0 Fax:+49 711 89661-300 sales.de@etas.com
ETAS Inc. Ann Arbor, MI 48103, USA Tel.: +1 888 ETAS INC Fax: +1 734 997 9449 sales.us@etas.com	**ETAS S.A.S.** 94588 Rungis Cedex, France Tel.: +33 (1) 56 70 00 50 Fax: +33 (1) 56 70 00 51 sales.fr@etas.com
ETAS K.K. Yokohama 220-6217, Japan Tel.: +81 45 222-0900 Fax: +81 45 222-0956 sales.jp@etas.com	**ETAS Ltd.** Derby DE21 4SU, UK Tel.: +44 1332 253770 Fax: +44 1332 253779 sales.uk@etas.com
ETAS Korea Co. Ltd. Seoul 137-889, Korea Tel.: +82 2 5747-016 Fax: +82 2 5747-120 sales.kr@etas.com	**ETAS (Shanghai) Co., Ltd.** Shanghai 200120, P.R. China Tel.: +86 21 5037 2220 Fax: +86 21 5037 2221 sales.cn@etas.com
ETAS in Italy 10135 TORINO, Italy Tel.: +39 011 3285 988 Fax: +39 (011) 3285 256 sales.it@etas.com	**ETAS Automotive India Pvt. Ltd.** Bangalore 560 068, India Tel.: +91 80 4191 2585 Fax: +91 80 4191 2586 sales.in@etas.com
ETAS in Brazil CEP-05802-140 São Paulo, Brazil Tel.: +55 11 2162-0252 sales.br@etas.com	**ETAS in Russia** Moscow, 129515, Russia Tel.: +7 495 937 0400 998 sales.ru@etas.com

Copyright Notice

Version: B00096-003

Disclaimer

The information in this document is subject to change without notice and does not represent a commitment on any part of ETAS. While the information contained herein is assumed to be accurate, ETAS assumes no responsibility for any errors or omissions.

In no event shall ETAS, 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 RTA-TRACE are trademarks of ETAS GmbH.

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Continental Automotive GmbH.

All other product names are trademarks or registered trademarks of their respective owners.

Safety Notice

This ETAS product fulfills standard quality management requirements. If requirements of specific safety standards (e.g. IEC 61508, ISO 26262) need to be fulfilled, these requirements must be explicitly defined and ordered by the customer. Before use of the product, customer must verify the compliance with specific safety standards.

Contents

1 About this Guide

This guide provides target-specific information for the TriCore/HighTec port of ETAS' RTA-OSEK. It supplements the more general information in the *RTA-OSEK User Guide*.

A port is defined as a specific target microcontroller/target toolchain pairing. This guide tells you about integration issues with your target toolchain and issues that you need to be aware of when using RTA-OSEK on your target hardware. Port specific parameters of implementation are also provided, giving the RAM and ROM requirements for each object in the RTA-OSEK Component and execution times for each API call to the RTA-OSEK Component.

1.1 Who Should Read this Guide?

The reader should have an understanding of real time embedded programming in an OSEK context. You should read this guide if you want to know low-level technical information to integrate the RTA-OSEK Component into your application.

1.2 Conventions

Important: Notes that appear like this contain important information that you need to be aware of. Make sure that you read them carefully and that you follow any instructions that you are given.

Portability: Notes that appear like this describe things that you will need to know if you want to write code that will work on any processor running the RTA-OSEK Component.

Program code, file names, C types and symbols, and RTA-OSEK API call names all appear in the `courier` typeface. When the name of an object is made available to the programmer the name also appears in the `courier` typeface, so, for example, a task named Task1 appears as a task handle called `Task1`.

2 Toolchain Issues

This chapter contains important details about RTA-OSEK and your toolchain. A port of the RTA-OSEK Component is specific to both the target hardware and a specific version of the compiler toolchain. You must make sure that you build your application with the supported toolchain.

If you are interested in using a different version of the same toolchain, please contact ETAS to confirm whether or not this is possible.

2.1 Compiler

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

Vendor	HighTec EDV-Systeme GmbH
Compiler	GNU TriCore C/C++ Compiler
Version	GNU C version 3.4.5 (Tool Version v2.15)

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

Option	Description
`-c`	The input file is compiled to a object file.
`-D%CPU_TYPE%`	Predefines name as a macro
`-I%CBASE_INC%\machine`	Includes the path for the compiler header files
`-mcpu=%CPU_TYPE%`	Select target CPU_TYPE. For example, tc1797, as in the example application

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`	The input file is compiled to a object file.
`-Os`	Optimize for size.
`-I%CBASE_INC%\machine`	Includes the path for the compiler header files
`-mcpu=%CPU_TYPE%`	Select target CPU_TYPE. For example, tc1797, as in the example application

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

Option	Description
-g	Generate debugging information
-msmall	Generates code for the big or small memory model. The small memory model assumed that all data fits into one 64K word page.

To support the use of multiple CPU configurations the environment variable CPU_TYPE should be set up to match the desired CPU target (e.g. tc1796).

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

2.2 Assembler

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

Vendor	HighTec EDV-Systeme GmbH
Assembler	GNU assembler
Version	version 2.13 (Tool Version v5.6)

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

Option	Description
-D%CPU_TYPE%	Predefines name as a macro

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

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_text	ROM	RTA-OSEK library code
.os_intvect	ROM	RTA-OSEK interrupt wrappers
.bss.os_pur	RAM	RTA-OSEK uninitialized data; zeroed by StartOS()
.bss.*.os_pur2	RAM	RTA-OSEK uninitialized data; must be zeroed during C-startup
.rodata.*.os_pid	ROM	RTA-OSEK read-only data
.rodata.*.os_pid2	ROM	RTA-OSEK read-only data
.rodata.os_pird	ROM	RTA-OSEK initialization data for os_pir
.rodata.os_pnird	ROM	RTA-OSEK initialization data for os_pnir

Sections	Rom/Ram	Description
.data.os_pir	RAM	RTA-OSEK initialized data; initialized by StartOS()
.data.*.os_pir2	RAM	RTA-OSEK initialized data; must be initialized during C-startup
.sdata.os_pnir	RAM	RTA-OSEK SDA initialized data; initialized by StartOS()
.sdata.os_pnid	RAM	RTA-OSEK SDA initialized data; initialized by StartOS()
.sdata.os_pnir2	RAM	RTA-OSEK SDA initialized data; must be initialized during C-startup
.sbss.os_pnur	RAM	RTA-OSEK SDA uninitialized data; must be zeroed during C-startup
.sbss.os_cntr	RAM	RTA-OSEK SDA uninitialized data; must be zeroed during C-startup
.bss.os_trace_ram	RAM	RTA-TRACE uninitialized data; must be zeroed during C-startup
.data.os_trace_ram	RAM	RTA-TRACE initialized data; must be zeroed during C-startup

It is not possible to compile osekdefs.c with 'os_' prefixed data section names. Instead the default names (.bss, .data, .rodata) are used, and these may be grouped by the linker control file into a 'non-OS' group and an 'OS' group. See the Example application tc1797_extram.x file as an illustration of how to do this.

In the table above, the * in section names may be empty, or a1, a2, a4 to denote data item alignment.

The RTA-OSEK Component requires the user stack to be quad word aligned. This is demonstrated in the linker directive file supplied with the example application.

Important: The RTA-OSEK Component makes use of relative 24-bit signed addressing mode. This means the library must be contained within a 16Mbyte memory block. 32-bit addressing is used externally providing no restrictions on placement of user code and data.

For improved performance, the RTA-OSEK Component uses a small amount of RAM data that should be located in a section that is addressable with a sign-extended 16-bit offset from address register A0 (i.e. the SDA). Please refer to the compiler documentation on the use of this section.

2.3.1 Direct Addressing

By default 32-bit addressing is used between application code and the RTA-OSEK Component so that the entire memory space can be supported.

The RTA-OSEK component can also use direct addressing to reduce the code size and improve performance. Direct addressing requires that all code objects must be located either within a relative signed 24-bit displacement address range of the current address, or an absolute disp24 address range (See the Infineon v1.3 Instruction set manual for further details). These address ranges cover either a ±16 Mbytes region from the current address, or the first 2 Mbytes of every 256 Mbytes. Direct addressing cannot be used between code objects located in the internal FLASH memory and the external memory regions or in the CSRAM of v1.3 CPUs. To make use of direct addressing application code should be compiled with the macro `OS_DIRECT_CALLS` defined.

All PC-relative jump and call instructions are subject to relaxation in order to find the smallest possible instruction (or instruction sequence) that can reach the specified target.The maximum allowed displacement for local branches is +/-16 MB. call, fcall, j and jl instructions branching to global functions can be relaxed by the linker to reach any valid code address within TriCore's entire 32-bit address space; In order to enable this feature, the option --relax-24rel (or -relax) is passed to tricore-ld.

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	Lauterbach Trace32

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for TriCore/HighTec. 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:

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:

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. Must be aligned to a power-of-two (>= 32) byte boundary, dependent on the range of priorities used. 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 HighTec EDV-Systeme GmbH C compiler can generate appropriate interrupt handling code for a C function decorated with the `__attribute__((interrupt_handler))` 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.s`.

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 (VV represents the 2 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
BIV + 0x0020	os_wrapper_01
BIV + 0x0040	os_wrapper_02
…	…
BIV + 0x1fe0	os_wrapper_ff

3.1.6 Vector Table Generation

If RTA-OSEK is used to generate the TriCore vector table, all the vectors are placed in a section `.os_intvect`. The Example application illustrates how to use the linker control file to define a symbol for the base of the vector table, and use this to initialize the `BIV` register.

The Category 2 interrupt wrappers will be generated in the vector table with a unique wrapper `os_wrapper_`*vv*, where *vv* corresponds with the vector number. They are placed into one section in the linker directive file, as follows:

```
.vecttab ALIGN(0x2000) :
         {
         *(.traptab)
         os_TriCore_vector_table = .;
         *(.os_intvect)
         } > ext_cram = 0
```

3.1.7 IO Privilege Mode

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

3.1.8 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.

3.1.9 Straightforward

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 vectors used does not need to be contiguous: there may be gaps in the range of vectors used.

The Category 1 ISRs would be written as

```
void D(void) __attribute__((interrupt_handler));
void D(void)
{
  /* handler code for D */
}
```

and

```
void C(void) __attribute__((interrupt_handler));
void C(void)
{
  /* handler code for C */
}
```

3.1.10 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.

3.1.11 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 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.		

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.

3.1.12 Default Interrupt

The default interrupt handler does not require any attributes 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	**Notes**
BIV	Interrupt vector table base address - must be set to the symbol defined in the linker control file
FCX	Free context list. This should be initialized to point to a suitably large linked list of context areas (performed by HighTec C startup code).
PSW.IO	Supervisor (0b10)

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

Register	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.

`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 details of the maximum stack usage within RTA-OSEK API calls is not yet available.

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

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 Initial Stack Pointer

The RTA-OSEK Component relies on the symbol `_os_empty_stack_sp` being defined with the initial value of the stack pointer. This can be achieved in the linker directive file, as in the following example:

```
.stack ((. + 7) & -8) (NOLOAD) :
{
  . = . + __USTACK_SIZE ;
  /* stack works down, so allocate space, then label
it */
  /* used by C start-up code */
```

```
  __USTACK = . ;
  _os_empty_stack_sp = __USTACK;          /* used by
RTA-OSEK as stack base */
} > int_dram
```

3.3.5 Context Save Areas (CSAs)

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

3.3.6 Call Depth Counter

The RTA-OSEK Component can operate with the TriCore call depth counter (PSW.CDC) 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	**TC1796**
Clock speed (MHz)	80
Code memory	Internal scratchpad RAM
Read-only data memory	External RAM
Read-write data memory	Internal scratchpad 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

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	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	46	46	46	46	46	46
	ROM	164	164	168	256	256	260
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	66	66	66	66	66	66
	ROM	236	236	240	328	328	332
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	84	84	84	84	84	84
	ROM	292	292	296	384	384	388
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	48	56	n/a	48	56
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	76	76	76	76	76	76
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	40	40	40	40	40	40
Counter	RAM	4	4	4	4	4	4
	ROM	108	108	108	108	108	108
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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	48	48	48	48	48	48
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		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	60	68	n/a	60	68
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	n/a	n/a	n/a	n/a	n/a	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
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	108	108	108	108	108	108
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	116	116	116	116	116	116
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	40	40	40	40	40	40
Counter	RAM	4	4	4	4	4	4
	ROM	108	108	108	108	108	108
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
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	48	48	48	48	48	48
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	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

Configuration Events Shared Task Priorities Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
	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 Events Shared Task Priorities Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		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	68	76	n/a	68	76
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	120	120	120	120	120	120
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	128	128	128	128	128	128
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Alarm	RAM	12	12	12	12	12	12
	ROM	44	44	44	44	44	44
Counter	RAM	4	4	4	4	4	4
	ROM	112	112	112	112	112	112
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
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
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	48	48	48	48	48	48
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	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.
Hook	Pre- and Post- Task hooks are used.
KL	API is called from OS level.
KL1i	API is called from OS level, idle task is only ECC task.
KL2	Activated taskset has one BCC2 task.
LExt	Used for lightweight termination in Extended Status.
ServiceID	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
Parameters	`ErrorHook` uses `GetServiceID` and `GetServiceParameters`.
NoHook	Pre- and/or Post- Task hooks are not used.
NS	No context switch is possible.
NS1i	No context switch is possible, idle task is only ECC task.
NS2	Activated taskset has one BCC2 task.
NSH	Chain from heavyweight task, not to higher priority.
NSL	Chain from lightweight task, not to higher priority.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.

Variant	Description
Task	Resource is used only by tasks.

Standard

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	134	180	220	142	186	254
	NS		118	166	204	126	172	238
	KL	2	68	116	154	76	122	188
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	130	180	216	138	186	250
	SWH	1, 9	160	206	242	168	212	276
	NSL	8	130	180	216	138	186	250
	NSH	9	154	200	236	162	206	270
Schedule			96	96	122	96	96	122
GetTaskID			32	32	32	32	32	32
GetTaskState			76	76	76	88	88	88
EnableAllInterrupts			28	28	28	28	28	28
DisableAllInterrupts			32	32	32	32	32	32
ResumeAllInterrupts			40	40	40	40	40	40
SuspendAllInterrupts			48	48	48	48	48	48
ResumeOSInterrupts			40	40	40	40	40	40
SuspendOSInterrupts			64	64	64	64	64	64
GetResource	Task	7	24	24	28	24	24	28
	Combined	6	70	70	70	70	70	70
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	70	70	70	70	70	70
	Combined	6	168	168	168	168	168	168
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	104	104	186
	NS		n/a	n/a	n/a	86	86	172
	NS1i	10	n/a	n/a	n/a	60	n/a	n/a
	KL	2	n/a	n/a	n/a	48	48	132
	KL1i	2, 10	n/a	n/a	n/a	22	n/a	n/a
ClearEvent			n/a	n/a	n/a	66	66	66

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	256	256	496
	fp	11	n/a	n/a	n/a	282	282	542
	1i	10	n/a	n/a	n/a	20	n/a	n/a
GetAlarmBase			60	60	60	60	60	60
GetAlarm			88	88	88	88	88	88
SetRelAlarm			378	378	378	378	378	378
SetAbsAlarm			426	426	426	426	426	426
CancelAlarm			78	78	78	78	78	78
InitCounter			54	54	54	54	54	54
GetCounterValue			68	68	68	68	68	68
GetScheduleTableStatus		34	58	80	80	58	80	80
NextScheduleTable		34	70	174	174	70	174	174
StartScheduleTable		34	104	160	160	104	160	160
StopScheduleTable		34	76	108	108	76	108	108
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		12	12	12	12	12	12
ScheduleTable expiry point	Final		34	34	34	34	34	34
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	28	28	28	28	28	28
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		68	68	68	68	68	68
	KL	2	32	32	32	32	32	32
osek_incr_counter			44	44	44	44	44	44
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			236	236	236	236	236	236
ShutdownOS	NoHook	12	38	38	38	38	38	38
	Hook	13	50	50	50	50	50	50
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			24	24	24	24	24	24
StopCOM			20	20	20	20	20	20
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	68	68	68	168	168	168

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	168	168	168	168	168	168
GetMessageResource			38	38	38	38	38	38
ReleaseMessageResource			34	34	34	34	34	34
GetMessageStatus			42	42	42	42	42	42
SendMessage	SW CCCA	1, 14	88	88	88	212	212	212
	SW CCCB	1, 15	198	198	198	212	212	212
	NS CCCA	14	88	88	88	212	212	212
	NS CCCB	15	198	198	198	212	212	212
	KL CCCA	2, 14	50	50	50	174	174	174
	KL CCCB	2, 15	160	160	160	174	174	174
main_dispatch	NoHook	12	140	140	182	140	140	182
	Hook	13	176	176	220	176	176	220
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	92	92	92	92	92	92
	B1HF	21	100	100	100	100	100	100
	B2LI	22	n/a	76	100	n/a	76	100
	B2LF	23	n/a	82	106	n/a	82	106
	B2HI	24	n/a	282	344	n/a	282	344
	B2HF	25	n/a	286	352	n/a	286	352
	E1HI	26	n/a	n/a	n/a	434	434	500
	E1HF	27	n/a	n/a	n/a	442	442	508
	E2HI	28	n/a	n/a	n/a	n/a	n/a	500
	E2HF	29	n/a	n/a	n/a	n/a	n/a	508
ErrorHook support		16	36	36	36	36	36	36
	ServiceID	17	46	46	46	46	46	46
	Parameters	18	68	68	68	68	68	68
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	130	248	290	144	276	338
	NS		116	234	276	130	262	324
	KL	2	66	184	226	80	210	272
ChainTaskset	SWL	1, 8	142	266	306	148	282	344
	SWH	1, 9	184	308	348	190	324	386
	NSL	8	142	266	306	148	282	344
	NSH	9	178	302	342	184	318	380
GetTasksetRef			8	8	8	8	8	8

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
MergeTaskset			50	50	50	50	50	50
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			52	52	52	52	52	52
TestSubTaskset			52	52	52	52	52	52
TestEquivalentTaskset			50	50	50	50	50	50
TickSchedule	SW	1	174	150	150	150	150	150
	NS		156	128	128	128	128	128
	KL	2	116	86	86	86	86	86
AdvanceSchedule	SW	1	174	140	140	140	140	140
	NS		156	122	122	122	122	122
	KL	2	116	82	82	82	82	82
StartSchedule			74	74	74	74	74	74
StopSchedule			62	62	62	62	62	62
GetScheduleStatus			84	84	84	84	84	84
GetScheduleValue			68	68	68	68	68	68
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			8	8	8	8	8	8
SetArrivalpointNext			8	8	8	8	8	8
TestArrivalpointWritable			34	34	34	34	34	34
GetExecutionTime			4	4	4	4	4	4
GetLargestExecutionTime			8	8	8	8	8	8
ResetLargestExecutionTime			4	4	4	4	4	4
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	134	180	220	142	186	254
	NS		118	166	204	126	172	238
	KL	2	68	116	154	76	122	188
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	130	180	216	138	186	250
	SWH	1, 9	160	206	242	168	212	276
	NSL	8	130	180	216	138	186	250
	NSH	9	154	200	236	162	206	270
Schedule			118	118	144	118	118	144
GetTaskID			32	32	32	32	32	32
GetTaskState			76	76	76	88	88	88
EnableAllInterrupts			28	28	28	28	28	28
DisableAllInterrupts			32	32	32	32	32	32
ResumeAllInterrupts			40	40	40	40	40	40
SuspendAllInterrupts			48	48	48	48	48	48
ResumeOSInterrupts			40	40	40	40	40	40
SuspendOSInterrupts			64	64	64	64	64	64
GetResource	Task	7	24	24	28	24	24	28
	Combined	6	70	70	70	70	70	70
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	94	94	94	94	94	94
	Combined	6	214	214	214	214	214	214
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	104	104	186
	NS		n/a	n/a	n/a	86	86	172
	NS1i	10	n/a	n/a	n/a	60	n/a	n/a
	KL	2	n/a	n/a	n/a	48	48	132
	KL1i	2, 10	n/a	n/a	n/a	22	n/a	n/a
ClearEvent			n/a	n/a	n/a	66	66	66
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	350	350	588
	fp	11	n/a	n/a	n/a	376	376	634

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	110	n/a	n/a
GetAlarmBase			60	60	60	60	60	60
GetAlarm			88	88	88	88	88	88
SetRelAlarm			378	378	378	378	378	378
SetAbsAlarm			426	426	426	426	426	426
CancelAlarm			78	78	78	78	78	78
InitCounter			54	54	54	54	54	54
GetCounterValue			68	68	68	68	68	68
GetScheduleTableStatus		34	58	80	80	58	80	80
NextScheduleTable		34	70	174	174	70	174	174
StartScheduleTable		34	104	160	160	104	160	160
StopScheduleTable		34	76	108	108	76	108	108
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		12	12	12	12	12	12
ScheduleTable expiry point	Final		34	34	34	34	34	34
GetISRID		4	30	30	30	30	30	30
Process container	Yielding	32	28	28	28	28	28	28
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		68	68	68	68	68	68
	KL	2	32	32	32	32	32	32
osek_incr_counter			44	44	44	44	44	44
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			302	302	302	302	302	302
ShutdownOS	NoHook	12	38	38	38	38	38	38
	Hook	13	50	50	50	50	50	50
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			24	24	24	24	24	24
StopCOM			20	20	20	20	20	20
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	68	68	68	168	168	168
	CCCB	15	168	168	168	168	168	168
GetMessageResource			38	38	38	38	38	38
ReleaseMessageResource			34	34	34	34	34	34

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			42	42	42	42	42	42
SendMessage	SW CCCA	1, 14	88	88	88	212	212	212
	SW CCCB	1, 15	198	198	198	212	212	212
	NS CCCA	14	88	88	88	212	212	212
	NS CCCB	15	198	198	198	212	212	212
	KL CCCA	2, 14	50	50	50	174	174	174
	KL CCCB	2, 15	160	160	160	174	174	174
main_dispatch	NoHook	12	184	184	222	184	184	222
	Hook	13	228	228	268	228	228	268
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	84	84	84	84	84	84
	B1HF	21	92	92	92	92	92	92
	B2LI	22	n/a	42	66	n/a	42	66
	B2LF	23	n/a	48	72	n/a	48	72
	B2HI	24	n/a	220	282	n/a	220	282
	B2HF	25	n/a	224	290	n/a	224	290
	E1HI	26	n/a	n/a	n/a	428	428	494
	E1HF	27	n/a	n/a	n/a	436	436	502
	E2HI	28	n/a	n/a	n/a	n/a	n/a	494
	E2HF	29	n/a	n/a	n/a	n/a	n/a	502
ErrorHook support		16	36	36	36	36	36	36
	ServiceID	17	46	46	46	46	46	46
	Parameters	18	68	68	68	68	68	68
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	88	88	88	88	88	88
ActivateTaskset	SW	1	130	248	290	144	276	338
	NS		116	234	276	130	262	324
	KL	2	66	184	226	80	210	272
ChainTaskset	SWL	1, 8	142	266	306	148	282	344
	SWH	1, 9	184	308	348	190	324	386
	NSL	8	142	266	306	148	282	344
	NSH	9	178	302	342	184	318	380
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			50	50	50	50	50	50
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			52	52	52	52	52	52

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			52	52	52	52	52	52
TestEquivalentTaskset			50	50	50	50	50	50
TickSchedule	SW	1	174	150	150	150	150	150
	NS		156	128	128	128	128	128
	KL	2	116	86	86	86	86	86
AdvanceSchedule	SW	1	174	140	140	140	140	140
	NS		156	122	122	122	122	122
	KL	2	116	82	82	82	82	82
StartSchedule			74	74	74	74	74	74
StopSchedule			62	62	62	62	62	62
GetScheduleStatus			84	84	84	84	84	84
GetScheduleValue			68	68	68	68	68	68
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			8	8	8	8	8	8
SetArrivalpointNext			8	8	8	8	8	8
TestArrivalpointWritable			34	34	34	34	34	34
GetExecutionTime			110	110	110	110	110	110
GetLargestExecutionTime			14	14	14	14	14	14
ResetLargestExecutionTime			12	12	12	12	12	12
GetStackOffset			18	18	18	18	18	18

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	226	280	320	236	286	348
	NS		272	324	362	282	330	390
	KL	2	128	180	218	138	186	246
TerminateTask	LExt	3	96	96	96	96	96	96

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	120	120	120	120	120	120
ChainTask	SWL	1, 8	244	304	340	252	310	372
	SWH	1, 9	288	336	374	298	342	400
	NSL	8	294	354	390	302	360	422
	NSH	9	332	380	418	342	386	444
Schedule			244	244	270	244	244	270
GetTaskID			48	48	48	48	48	48
GetTaskState			226	226	226	222	222	222
EnableAllInterrupts			44	44	44	44	44	44
DisableAllInterrupts			48	48	48	48	48	48
ResumeAllInterrupts			80	80	80	80	80	80
SuspendAllInterrupts			64	64	64	64	64	64
ResumeOSInterrupts			80	80	80	80	80	80
SuspendOSInterrupts			80	80	80	80	80	80
GetResource	Task	7	342	342	306	342	342	306
	Combined	6	316	316	316	316	316	316
	CLEx	3	268	268	268	268	268	268
ReleaseResource	Task	7	322	322	322	322	322	322
	Combined	6	430	430	430	430	430	430
	CLEx	3	280	280	280	280	280	280
SetEvent	SW	1	n/a	n/a	n/a	270	270	354
	NS		n/a	n/a	n/a	308	308	390
	NS1i	10	n/a	n/a	n/a	220	n/a	n/a
	KL	2	n/a	n/a	n/a	172	172	254
	KL1i	2, 10	n/a	n/a	n/a	134	n/a	n/a
ClearEvent			n/a	n/a	n/a	130	130	130
GetEvent			n/a	n/a	n/a	124	124	124
WaitEvent	<default>		n/a	n/a	n/a	444	444	672
	fp	11	n/a	n/a	n/a	470	470	718
	1i	10	n/a	n/a	n/a	208	n/a	n/a
GetAlarmBase			192	192	192	192	192	192
GetAlarm			166	166	166	166	166	166
SetRelAlarm			500	500	500	500	500	500
SetAbsAlarm			532	532	532	532	532	532
CancelAlarm			154	154	154	154	154	154
InitCounter			204	204	204	204	204	204
GetCounterValue			198	198	198	198	198	198

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	74	96	96	74	96	96
NextScheduleTable		34	86	190	190	86	190	190
StartScheduleTable		34	120	176	176	120	176	176
StopScheduleTable		34	92	124	124	92	124	124
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		12	12	12	12	12	12
ScheduleTable expiry point	Final		34	34	34	34	34	34
GetISRID		4	46	46	46	46	46	46
Process container	Yielding	32	28	28	28	28	28	28
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		110	110	110	110	110	110
	KL	2	32	32	32	32	32	32
osek_incr_counter			44	44	44	44	44	44
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			318	318	318	318	318	318
ShutdownOS	NoHook	12	48	48	48	48	48	48
	Hook	13	60	60	60	60	60	60
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			40	40	40	40	40	40
StopCOM			44	44	44	44	44	44
ReadFlag			32	32	32	32	32	32
ResetFlag			36	36	36	36	36	36
ReceiveMessage	CCCA	14	168	168	168	272	272	272
	CCCB	15	272	272	272	272	272	272
GetMessageResource			80	80	80	80	80	80
ReleaseMessageResource			78	78	78	78	78	78
GetMessageStatus			84	84	84	84	84	84
SendMessage	SW CCCA	1, 14	206	206	206	332	332	332
	SW CCCB	1, 15	318	318	318	332	332	332
	NS CCCA	14	206	206	206	332	332	332
	NS CCCB	15	318	318	318	332	332	332
	KL CCCA	2, 14	124	124	124	250	250	250
	KL CCCB	2, 15	236	236	236	250	250	250
main_dispatch	NoHook	12	184	184	222	184	184	222

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	228	228	268	228	228	268
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	84	84	84	84	84	84
	B1HF	21	92	92	92	92	92	92
	B2LI	22	n/a	42	66	n/a	42	66
	B2LF	23	n/a	48	72	n/a	48	72
	B2HI	24	n/a	220	282	n/a	220	282
	B2HF	25	n/a	224	290	n/a	224	290
	E1HI	26	n/a	n/a	n/a	428	428	494
	E1HF	27	n/a	n/a	n/a	436	436	502
	E2HI	28	n/a	n/a	n/a	n/a	n/a	494
	E2HF	29	n/a	n/a	n/a	n/a	n/a	502
ErrorHook support		16	124	124	124	124	124	124
	ServiceID	17	132	132	132	132	132	132
	Parameters	18	154	154	154	154	154	154
validity_checks		3	22	22	22	22	22	22
Timing_dispatch		4	80	80	80	80	80	80
Timing_termination		4	88	88	88	88	88	88
ActivateTaskset	SW	1	294	384	422	328	410	464
	NS		328	420	462	364	448	504
	KL	2	198	286	328	238	314	370
ChainTaskset	SWL	1, 8	336	424	464	362	438	496
	SWH	1, 9	384	476	516	412	488	548
	NSL	8	376	468	510	406	484	540
	NSH	9	418	512	552	450	526	584
GetTasksetRef			106	106	106	106	106	106
MergeTaskset			264	264	264	264	264	264
AssignTaskset			154	154	154	154	154	154
RemoveTaskset			264	264	264	264	264	264
TestSubTaskset			280	280	280	280	280	280
TestEquivalentTaskset			278	278	278	278	278	278
TickSchedule	SW	1	296	258	258	258	258	258
	NS		328	320	320	320	320	320
	KL	2	220	186	186	186	186	186
AdvanceSchedule	SW	1	304	266	266	266	266	266
	NS		336	322	322	322	322	322
	KL	2	228	192	192	192	192	192

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			238	238	238	238	238	238
StopSchedule			198	198	198	198	198	198
GetScheduleStatus			222	222	222	222	222	222
GetScheduleValue			184	184	184	184	184	184
GetScheduleNext			84	84	84	84	84	84
SetScheduleNext			160	160	160	160	160	160
GetArrivalpointDelay			116	116	116	116	116	116
SetArrivalpointDelay			136	136	136	136	136	136
GetArrivalpointTasksetRef			112	112	112	112	112	112
GetArrivalpointNext			116	116	116	116	116	116
SetArrivalpointNext			166	166	166	166	166	166
TestArrivalpointWritable			130	130	130	130	130	130
GetExecutionTime			168	168	168	168	168	168
GetLargestExecutionTime			96	96	96	96	96	96
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	114	235	330	115	177	319
	NS	102	224	319	105	167	308
	KL	92	213	309	95	156	297
TerminateTask	LExt	0	0	0	0	0	0
	H	123	124	126	125	124	124
ChainTask	SWL	319	467	664	436	513	756
	SWH	394	506	704	514	553	796
	NSL	320	467	664	437	513	756
	NSH	393	506	704	510	552	795
Schedule	SW	105	104	127	104	104	127
GetTaskID		36	36	36	36	36	38
GetTaskState		63	62	62	67	67	67
EnableAllInterrupts		20	20	20	20	20	20
DisableAllInterrupts		16	16	16	16	16	16
ResumeAllInterrupts		23	23	24	23	23	24
SuspendAllInterrupts		19	19	19	20	19	19
ResumeOSInterrupts		23	23	23	23	23	25
SuspendOSInterrupts		20	20	19	20	19	19
GetResource	Task	83	83	84	83	83	84
	Combined	93	94	93	94	94	93
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	46	45	46	46	45	46
	Combined	91	93	92	91	93	92
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	119	119	119
	NS	n/a	n/a	n/a	86	85	85
	KL	n/a	n/a	n/a	76	76	76
ClearEvent		n/a	n/a	n/a	43	43	42
GetEvent		n/a	n/a	n/a	30	30	30
WaitEvent	<default>	n/a	n/a	n/a	448	449	570
	fp	n/a	n/a	n/a	471	470	592
GetAlarmBase		82	81	95	81	96	81
GetAlarm		83	82	81	82	81	81

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		110	109	109	108	110	108
SetAbsAlarm		108	108	107	108	110	106
CancelAlarm		45	43	45	44	46	44
InitCounter		59	58	58	59	59	59
GetCounterValue		60	61	61	60	63	60
osek_tick_alarm	<default>	47	48	49	47	48	48
	KL	37	37	37	37	37	37
osek_incr_counter		22	22	23	22	23	22
GetActiveApplicationMode		4	4	4	4	4	4
StartOS		1439	1438	1440	1438	1438	1439
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	19	19	17	18	19	18
InitCOM		8	8	8	8	8	8
CloseCOM		7	7	7	7	7	7
StartCOM		25	25	25	56	56	56
StopCOM		11	11	11	12	11	11
ReadFlag		n/a	n/a	n/a	26	26	26
ResetFlag		n/a	n/a	n/a	24	24	24
ReceiveMessage		41	41	41	235	238	236
GetMessageResource		n/a	n/a	n/a	160	160	161
ReleaseMessageResource		n/a	n/a	n/a	133	133	134
GetMessageStatus		n/a	n/a	n/a	57	57	57
SendMessage	SW	212	333	429	424	485	628
	NS	203	326	421	415	479	619
	KL	181	302	398	394	455	596
ActivateTaskset	SW	39	326	459	39	308	466
	NS	29	316	451	31	299	489
	KL	19	307	439	19	260	478
	SW2	39	326	459	39	308	466
	NS2	29	316	451	31	299	489
	KL2	19	307	439	19	260	478
ChainTaskset	SWL	285	578	830	399	633	972
	SWH	360	654	908	473	739	1018
	NSL	284	577	861	399	663	971
	NSH	357	652	908	473	709	1051
GetTasksetRef		28	28	28	28	29	28

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
MergeTaskset		20	21	20	21	20	20
AssignTaskset		9	9	9	9	9	9
RemoveTaskset		20	20	21	21	21	20
TestSubTaskset		25	25	25	25	25	26
TestEquivalentTaskset		24	24	25	25	24	25
TickSchedule	SW	181	503	636	199	495	699
	NS	172	493	625	190	485	691
	KL	159	481	614	178	473	680
	SW2	181	503	636	199	457	677
	NS2	172	493	625	190	447	669
	KL2	159	481	614	178	435	658
AdvanceSchedule	SW	142	464	595	159	456	660
	NS	134	454	586	152	447	651
	KL	123	443	575	140	436	639
	SW2	142	464	595	159	418	638
	NS2	134	454	586	152	409	629
	KL2	123	443	575	140	398	617
StartSchedule		86	86	86	85	86	85
StopSchedule		84	84	84	85	85	86
GetScheduleStatus		85	86	85	85	86	85
GetScheduleValue		68	68	68	68	68	70
GetScheduleNext		30	30	30	30	31	30
SetScheduleNext		27	27	27	28	27	27
GetArrivalpointDelay		29	28	28	28	28	28
SetArrivalpointDelay		8	8	8	8	8	8
GetArrivalpointTasksetRef		10	10	10	10	10	10
GetArrivalpointNext		11	11	11	13	11	11
SetArrivalpointNext		8	8	9	8	8	8
TestArrivalpointWritable		18	18	18	19	18	19
GetExecutionTime		6	6	6	6	6	6
GetLargestExecutionTime		28	28	28	28	28	28
ResetLargestExecutionTime		8	8	8	8	8	8
GetStackOffset		13	13	13	13	13	13

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	113	233	331	115	177	319
	NS	102	224	319	103	167	308
	KL	92	213	308	93	156	297
TerminateTask	LExt	0	0	0	0	0	0
	H	330	330	330	328	331	333
ChainTask	SWL	592	738	934	708	789	1032
	SWH	681	792	989	797	842	1086
	NSL	593	738	935	707	788	1033
	NSH	679	790	987	794	841	1084
Schedule	SW	104	105	126	104	104	126
GetTaskID		36	37	35	35	36	36
GetTaskState		63	62	62	68	68	67
EnableAllInterrupts		20	20	19	19	21	21
DisableAllInterrupts		16	16	15	14	16	17
ResumeAllInterrupts		25	23	22	23	23	23
SuspendAllInterrupts		19	19	17	18	18	19
ResumeOSInterrupts		24	23	22	22	23	23
SuspendOSInterrupts		19	19	17	18	19	20
GetResource	Task	83	83	83	82	83	84
	Combined	93	94	93	93	94	94
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	45	45	42	44	46	45
	Combined	93	92	91	91	92	91
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	118	120	120
	NS	n/a	n/a	n/a	83	86	84
	KL	n/a	n/a	n/a	75	76	75
ClearEvent		n/a	n/a	n/a	40	43	43
GetEvent		n/a	n/a	n/a	29	30	30
WaitEvent	<default>	n/a	n/a	n/a	632	633	758
	fp	n/a	n/a	n/a	655	655	779
GetAlarmBase		96	96	82	94	81	95
GetAlarm		82	82	81	80	81	81

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		109	109	109	108	109	110
SetAbsAlarm		108	108	108	107	107	107
CancelAlarm		45	45	45	42	44	44
InitCounter		58	58	59	57	58	59
GetCounterValue		61	61	61	59	61	61
osek_tick_alarm	<default>	47	47	47	47	48	49
	KL	37	37	36	36	37	37
osek_incr_counter		23	22	22	21	22	22
GetActiveApplicationMode		4	4	3	3	4	4
StartOS		4522	4521	4518	4518	4519	4518
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	17	17	17	17	20	18
InitCOM		8	8	7	7	8	8
CloseCOM		7	7	6	6	7	7
StartCOM		25	25	24	57	56	57
StopCOM		11	11	10	10	11	12
ReadFlag		n/a	n/a	n/a	25	26	26
ResetFlag		n/a	n/a	n/a	23	24	24
ReceiveMessage		42	42	40	234	237	236
GetMessageResource		n/a	n/a	n/a	159	160	160
ReleaseMessageResource		n/a	n/a	n/a	133	133	134
GetMessageStatus		n/a	n/a	n/a	56	57	57
SendMessage	SW	212	332	430	424	485	628
	NS	203	325	420	414	478	619
	KL	181	302	397	393	455	596
ActivateTaskset	SW	38	324	427	37	276	497
	NS	29	347	419	29	301	458
	KL	19	306	439	17	288	478
	SW2	38	324	427	37	276	497
	NS2	29	347	419	29	301	458
	KL2	19	306	439	17	288	478
ChainTaskset	SWL	557	818	1131	670	937	1281
	SWH	647	938	1159	758	1028	1369
	NSL	557	849	1103	669	909	1250
	NSH	644	940	1188	756	1028	1369
GetTasksetRef		28	28	27	27	29	29

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		20	20	19	20	21	20
AssignTaskset		9	9	8	8	9	9
RemoveTaskset		20	20	20	19	21	21
TestSubTaskset		25	25	25	24	25	25
TestEquivalentTaskset		24	24	23	23	24	25
TickSchedule	SW	181	503	636	198	523	699
	NS	172	492	627	189	513	687
	KL	160	481	614	177	502	676
	SW2	181	503	636	198	486	677
	NS2	172	492	627	189	476	665
	KL2	160	481	614	177	465	654
AdvanceSchedule	SW	142	463	596	158	484	658
	NS	134	454	586	148	474	648
	KL	123	442	576	137	464	637
	SW2	142	463	596	158	447	636
	NS2	134	454	586	148	437	626
	KL2	123	442	576	137	427	615
StartSchedule		86	86	86	84	85	85
StopSchedule		85	84	83	85	85	84
GetScheduleStatus		86	85	84	84	86	85
GetScheduleValue		69	68	68	68	68	68
GetScheduleNext		30	30	29	29	30	30
SetScheduleNext		27	27	26	26	27	27
GetArrivalpointDelay		28	28	27	27	28	28
SetArrivalpointDelay		8	8	7	7	8	8
GetArrivalpointTasksetRef		11	11	9	9	10	10
GetArrivalpointNext		11	11	11	10	11	11
SetArrivalpointNext		9	8	7	7	8	8
TestArrivalpointWritable		18	18	17	17	18	18
GetExecutionTime		83	83	82	82	84	84
GetLargestExecutionTime		64	64	63	63	64	64
ResetLargestExecutionTime		44	44	43	43	45	44
GetStackOffset		10	10	9	9	10	10

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	663	802	901	669	745	886
	NS	776	916	1014	780	860	999
	KL	633	772	868	638	718	855
TerminateTask	LExt	360	361	363	363	361	361
	H	441	443	446	446	445	444
ChainTask	SWL	1225	1391	1588	1343	1444	1683
	SWH	1312	1442	1640	1432	1495	1735
	NSL	1346	1510	1710	1461	1562	1802
	NSH	1432	1561	1759	1552	1614	1851
Schedule	SW	151	150	173	151	151	173
GetTaskID		40	39	40	41	39	39
GetTaskState		685	686	686	670	668	670
EnableAllInterrupts		23	22	23	23	22	23
DisableAllInterrupts		17	19	19	19	17	20
ResumeAllInterrupts		29	28	31	29	30	29
SuspendAllInterrupts		21	21	21	23	20	23
ResumeOSInterrupts		29	28	30	29	29	29
SuspendOSInterrupts		21	21	21	23	20	24
GetResource	Task	1348	1347	552	1474	1473	679
	Combined	442	440	440	567	567	567
	CLEx	486	486	486	612	612	612
ReleaseResource	Task	425	423	424	552	550	552
	Combined	372	373	374	499	500	502
	CLEx	420	418	419	546	545	545
SetEvent	SW	n/a	n/a	n/a	717	714	716
	NS	n/a	n/a	n/a	727	727	727
	KL	n/a	n/a	n/a	662	663	661
ClearEvent		n/a	n/a	n/a	73	73	74
GetEvent		n/a	n/a	n/a	650	650	650
WaitEvent	<default>	n/a	n/a	n/a	735	734	835
	fp	n/a	n/a	n/a	756	756	856
GetAlarmBase		384	398	426	400	384	443
GetAlarm		380	379	421	379	379	422

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		451	450	494	452	450	492
SetAbsAlarm		426	425	469	428	429	469
CancelAlarm		343	341	385	342	342	384
InitCounter		742	741	804	741	740	805
GetCounterValue		319	320	322	320	321	321
osek_tick_alarm	<default>	61	61	61	62	62	63
	KL	34	34	35	35	34	35
osek_incr_counter		21	21	22	22	21	22
GetActiveApplicationMode		3	3	4	4	4	4
StartOS		4675	4674	4675	4674	4674	4675
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	19	19	21	20	19	20
InitCOM		7	7	8	8	7	8
CloseCOM		6	6	7	7	6	7
StartCOM		33	32	33	65	64	65
StopCOM		19	19	20	19	18	20
ReadFlag		n/a	n/a	n/a	35	34	35
ResetFlag		n/a	n/a	n/a	35	34	35
ReceiveMessage		217	219	218	414	414	413
GetMessageResource		n/a	n/a	n/a	881	881	881
ReleaseMessageResource		n/a	n/a	n/a	766	768	767
GetMessageStatus		n/a	n/a	n/a	221	221	221
SendMessage	SW	944	1084	1184	1160	1238	1378
	NS	1059	1199	1297	1272	1353	1491
	KL	895	1033	1129	1110	1191	1326
ActivateTaskset	SW	301	716	817	310	631	839
	NS	384	767	899	422	712	858
	KL	310	691	794	317	610	784
	SW2	301	716	817	310	631	839
	NS2	384	769	899	422	712	858
	KL2	310	691	794	317	606	784
ChainTaskset	SWL	975	1327	1611	1098	1352	1679
	SWH	1097	1453	1643	1189	1475	1800
	NSL	1057	1448	1673	1181	1472	1800
	NSH	1148	1508	1730	1272	1562	1887
GetTasksetRef		610	610	611	611	612	611

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
MergeTaskset		69	69	69	68	69	69
AssignTaskset		42	42	44	43	42	43
RemoveTaskset		66	66	67	66	68	66
TestSubTaskset		74	75	75	76	74	76
TestEquivalentTaskset		73	73	75	74	74	74
TickSchedule	SW	264	943	1047	553	934	1097
	NS	364	1041	1146	653	1036	1195
	KL	232	916	1019	526	906	1070
	SW2	264	943	1047	553	860	1037
	NS2	364	1041	1146	653	962	1135
	KL2	232	916	1019	526	832	1010
AdvanceSchedule	SW	226	911	1016	519	900	1064
	NS	328	1009	1112	622	999	1165
	KL	216	905	1006	515	893	1058
	SW2	226	911	1017	519	826	1004
	NS2	328	1009	1112	622	925	1105
	KL2	216	905	1006	515	819	1000
StartSchedule		156	153	154	157	157	154
StopSchedule		111	112	111	110	109	110
GetScheduleStatus		113	112	114	113	113	114
GetScheduleValue		96	95	97	96	95	94
GetScheduleNext		42	42	43	43	42	43
SetScheduleNext		95	95	95	95	94	96
GetArrivalpointDelay		49	49	51	50	49	50
SetArrivalpointDelay		55	54	56	56	54	56
GetArrivalpointTasksetRef		25	25	27	27	25	26
GetArrivalpointNext		28	28	28	28	27	29
SetArrivalpointNext		63	63	64	65	63	64
TestArrivalpointWritable		31	32	31	31	30	32
GetExecutionTime		97	97	99	98	98	99
GetLargestExecutionTime		607	607	608	608	608	607
ResetLargestExecutionTime		586	586	588	587	587	586
GetStackOffset		9	9	10	10	9	10

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	28	28	28	28	28	28
	Cat 2	29	42	42	42	42	42

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	28	28	28	28	28	28
	Cat 2	237	248	248	248	247	247

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	28	28	28	28	28	28
	Cat 2	237	247	248	249	248	248

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.

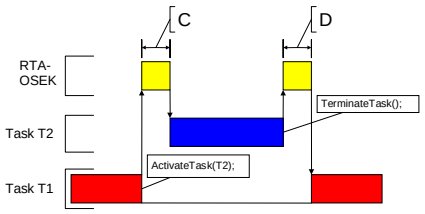

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

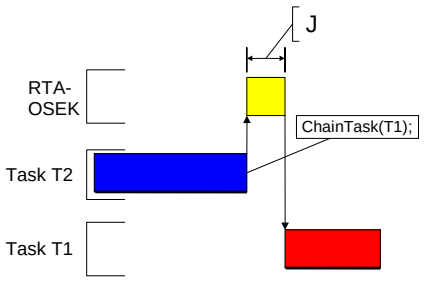

Figure 2: Task Chaining

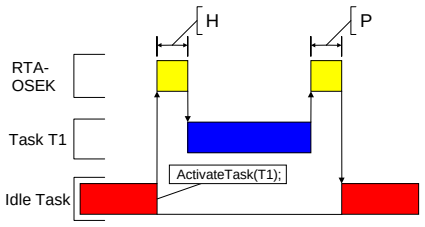

Figure 3: Task Activation from Idle Task

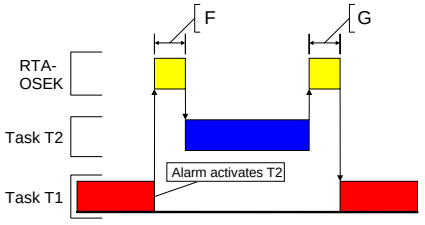

Figure 4: Task Activation from an Alarm

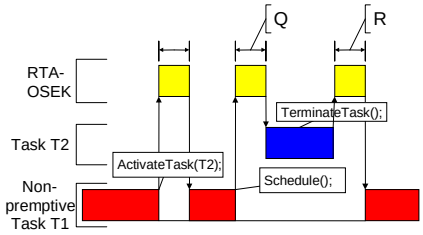

Figure 5: Non-Premptive Task Calls Schedule()

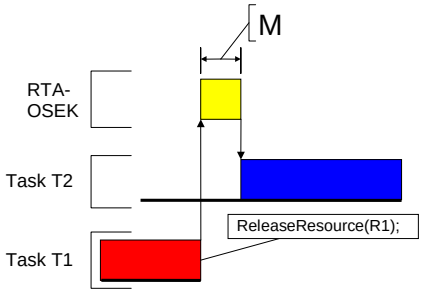

Figure 6: Blocked Task Activated by ReleaseResource()

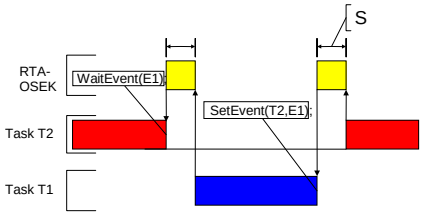

Figure 7: Waiting Task Activated by SetEvent()

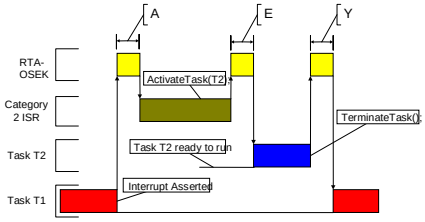

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	69	107	167	70	107	166
Figure 1: D	Heavy, Basic/Extended	122	157	220	194	195	254
ChainTask	Light, Basic	259	426	623	264	427	669
Figure 2: J	Heavy, Basic/Extended	478	644	905	556	681	985
Pre-emption	Light, Basic	201	349	568	205	348	613
Figure 1: C	Heavy, Basic/Extended	279	408	627	396	455	720
From idle task	Light, Basic	199	347	566	202	345	612
Figure 3: H	Heavy, Basic/Extended	277	406	625	393	452	719
Triggered by alarm	Light, Basic	237	386	606	240	384	652
Figure 4: F	Heavy, Basic/Extended	316	446	666	432	493	760
Schedule	Light, Basic	156	183	328	156	181	328
Figure 5: Q	Heavy, Basic/Extended	234	241	387	347	345	492
Release resource	Light, Basic	176	203	328	177	202	327
Figure 6: M	Heavy, Basic/Extended	254	262	388	368	366	491
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	500	501	788
From category 2 ISR	Light, Basic	124	161	283	135	159	284
Figure 8: E	Heavy, Basic/Extended	203	221	343	327	324	448

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	261	290	353	260	292	355
Figure 1: D	Heavy, Basic/Extended	330	355	415	392	393	455
ChainTask	Light, Basic	552	697	890	554	698	941
Figure 2: J	Heavy, Basic/Extended	989	1122	1379	1055	1160	1465
Pre-emption	Light, Basic	422	548	763	426	549	810
Figure 1: C	Heavy, Basic/Extended	480	609	824	597	658	920
From idle task	Light, Basic	420	546	762	424	547	808
Figure 3: H	Heavy, Basic/Extended	478	607	823	595	656	918
Triggered by alarm	Light, Basic	458	584	801	463	586	848
Figure 4: F	Heavy, Basic/Extended	516	646	863	634	695	958
Schedule	Light, Basic	376	382	523	378	383	524
Figure 5: Q	Heavy, Basic/Extended	434	443	584	549	549	691
Release resource	Light, Basic	398	403	522	399	405	524
Figure 6: M	Heavy, Basic/Extended	456	464	583	570	571	691
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	632	636	922
From category 2 ISR	Light, Basic	567	584	703	580	585	705
Figure 8: E	Heavy, Basic/Extended	624	644	762	749	751	872

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	359	390	454	363	391	452
Figure 1: D	Heavy, Basic/Extended	441	468	530	507	507	565
ChainTask	Light, Basic	1186	1349	1547	1189	1353	1594
Figure 2: J	Heavy, Basic/Extended	1733	1888	2146	1803	1930	2227
Pre-emption	Light, Basic	967	1110	1329	971	1111	1371
Figure 1: C	Heavy, Basic/Extended	1025	1170	1389	1142	1220	1481
From idle task	Light, Basic	966	1109	1328	970	1109	1370
Figure 3: H	Heavy, Basic/Extended	1024	1169	1388	1141	1219	1480
Triggered by alarm	Light, Basic	1021	1164	1382	1025	1165	1425
Figure 4: F	Heavy, Basic/Extended	1079	1225	1442	1196	1274	1535
Schedule	Light, Basic	418	421	566	416	422	565
Figure 5: Q	Heavy, Basic/Extended	476	481	626	587	588	732
Release resource	Light, Basic	730	735	857	855	861	982
Figure 6: M	Heavy, Basic/Extended	788	795	917	1026	1027	1149
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1237	1236	1526
From category 2 ISR	Light, Basic	585	599	722	596	599	720
Figure 8: E	Heavy, Basic/Extended	642	658	781	765	765	887

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	0	0	0	0	0	0
BCC1 lightweight, floating-point	0	0	0	0	0	0
BCC1 heavyweight, integer	0	0	0	0	0	0
BCC1 heavyweight, floating-point	16	16	16	16	16	16
BCC2 lightweight, integer	n/a	0	0	n/a	0	0
BCC2 lightweight, floating-point	n/a	0	0	n/a	0	0
BCC2 heavyweight, integer	n/a	16	16	n/a	16	16
BCC2 heavyweight, floating-point	n/a	16	16	n/a	16	16
ECC1 heavyweight, integer	n/a	n/a	n/a	18	18	16
ECC1 heavyweight, floating-point	n/a	n/a	n/a	18	18	16
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	16
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	16
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	0	0	0	0	0	0
BCC1 lightweight, floating-point	0	0	0	0	0	0
BCC1 heavyweight, integer	0	0	0	0	0	0
BCC1 heavyweight, floating-point	16	16	16	16	16	16
BCC2 lightweight, integer	n/a	0	0	n/a	0	0
BCC2 lightweight, floating-point	n/a	0	0	n/a	0	0
BCC2 heavyweight, integer	n/a	16	16	n/a	16	16
BCC2 heavyweight, floating-point	n/a	16	16	n/a	16	16
ECC1 heavyweight, integer	n/a	n/a	n/a	16	16	16
ECC1 heavyweight, floating-point	n/a	n/a	n/a	16	16	16
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	16
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	16

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		0	0	0	0	0	0
BCC1 lightweight, floating-point		0	0	0	0	0	0
BCC1 heavyweight, integer		16	16	16	16	16	16
BCC1 heavyweight, floating-point		16	16	16	16	16	16
BCC2 lightweight, integer		n/a	0	0	n/a	0	0
BCC2 lightweight, floating-point		n/a	0	0	n/a	0	0
BCC2 heavyweight, integer		n/a	16	16	n/a	16	16
BCC2 heavyweight, floating-point		n/a	16	16	n/a	16	16
ECC1 heavyweight, integer		n/a	n/a	n/a	18	18	16
ECC1 heavyweight, floating-point		n/a	n/a	n/a	18	18	16
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	16
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	16
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		0	0	0	0	0	0
BCC1 lightweight, floating-point		0	0	0	0	0	0
BCC1 heavyweight, integer		16	16	16	16	16	16
BCC1 heavyweight, floating-point		16	16	16	16	16	16
BCC2 lightweight, integer		n/a	0	0	n/a	0	0
BCC2 lightweight, floating-point		n/a	0	0	n/a	0	0
BCC2 heavyweight, integer		n/a	16	16	n/a	16	16
BCC2 heavyweight, floating-point		n/a	16	16	n/a	16	16
ECC1 heavyweight, integer		n/a	n/a	n/a	16	16	16
ECC1 heavyweight, floating-point		n/a	n/a	n/a	16	16	16
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	16
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	16

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	0	0	0	0	0	0
BCC1 lightweight, floating-point	0	0	0	0	0	0
BCC1 heavyweight, integer	16	16	16	16	16	16
BCC1 heavyweight, floating-point	16	16	16	16	16	16
BCC2 lightweight, integer	n/a	0	0	n/a	0	0
BCC2 lightweight, floating-point	n/a	0	0	n/a	0	0
BCC2 heavyweight, integer	n/a	16	16	n/a	16	16
BCC2 heavyweight, floating-point	n/a	16	16	n/a	16	16
ECC1 heavyweight, integer	n/a	n/a	n/a	26	26	16
ECC1 heavyweight, floating-point	n/a	n/a	n/a	26	26	16
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	16
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	16
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	0	0	0	0	0	0
BCC1 lightweight, floating-point	0	0	0	0	0	0
BCC1 heavyweight, integer	16	16	16	16	16	16
BCC1 heavyweight, floating-point	16	16	16	16	16	16
BCC2 lightweight, integer	n/a	0	0	n/a	0	0
BCC2 lightweight, floating-point	n/a	0	0	n/a	0	0
BCC2 heavyweight, integer	n/a	16	16	n/a	16	16
BCC2 heavyweight, floating-point	n/a	16	16	n/a	16	16
ECC1 heavyweight, integer	n/a	n/a	n/a	16	16	16
ECC1 heavyweight, floating-point	n/a	n/a	n/a	16	16	16
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	16
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	16

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for TriCore/HighTec supports two variations of the OSEK interrupt handling API calls. In addition to the API calls contained within the RTA-OSEK run-time libraries, inline versions are also supported. Using these inline versions will result in faster code and a reduced Context Save Area usage. The inline versions of these API calls all have the "os" prefix.

The inline API calls are restricted to the standard build applications that do not use RTA-TRACE. Inline calls contained within application code for other configurations will be automatically substituted with calls to the library API during compilation.

To take advantage of the inline versions in application code, the following substitutions should be used:

Library API call	Inline API call
`DisableAllInterrupts()`	`osDisableAllInterrupts()`
`EnableAllInterrupts()`	`osEnableAllInterrupts()`
`SuspendOSInterrupts()`	`osSuspendOSInterrupts()`
`ResumeOSInterrupts()`	`osResumeOSInterrupts()`
`SuspendAllInterrupts()`	`osSuspendAllInterrupts()`
`esumeAllInterrupts()`	`osResumeAllInterrupts()`

Support

For product support, please contact your local ETAS representative.

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