RTA-OSEK

Binding Manual: TriCore/GreenHills

Contact Details

ETAS Group

www.etasgroup.com

ETAS GmbH
70469 Stuttgart, Germany

Tel.:+49 711 89661-0
Fax:+49 711 89661-300

sales.de@etas.com

ETAS Inc.
Ann Arbor, MI 48103, USA

Tel.: +1 888 ETAS INC
Fax: +1 734 997 9449

sales.us@etas.com

ETAS K.K.
Yokohama 220-6217, Japan

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

sales.jp@etas.com

ETAS S.A.S.
94588 Rungis Cedex, France

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

sales.fr@etas.com

ETAS Korea Co. Ltd.
Seoul 137-889, Korea

Tel.: +82 2 5747-016
Fax: +82 2 5747-120

sales.kr@etas.com

ETAS Ltd.
Burton-upon-Trent
Staffordshire DE14 2WQ, UK

Tel.: +44 1283 54 65 12
Fax: +44 1283 54 87 67

sales.uk@etas.com

ETAS (Shanghai) Co., Ltd.
Shanghai 200120, P.R. China

Tel.: +86 21 5037 2220
Fax: +86 21 5037 2221

sales.cn@etas.com

ETAS Automotive India Pvt. Ltd.
Bangalore 560 068, India

Tel.: +91 80 4191 2585
Fax: +91 80 4191 2586

sales.in@etas.com

Copyright Notice

Version: B00084-002

Disclaimer

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

In no event shall LiveDevices, its employees, its contractors or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees or expenses of any nature or kind.

Trademarks

RTA-OSEK, RTA-TRACE and LiveDevices are trademarks of LiveDevices Ltd.

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

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

Contents

1 About this Guide

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

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

1.1 Who Should Read this Guide?

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

1.2 Conventions

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

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

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

2 Toolchain Issues

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

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

2.1 Compiler

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

Vendor	Green Hills Software, Inc.
Compiler	CCTRI
Version	MULTI v5.1.3 TriCore (Patch A)

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

Option	Description
`-cpu=<cpu>`	Select target CPU

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

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

Option	Description
`-cpu=<cpu>`	Select target CPU

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

Option	Description
`-g`	Generate debugging information

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 Green Hills 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	Green Hills Software, Inc.
Assembler	ASRTI
Version	MULTI v5.1.3 TriCore (Patch A)

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

Option	Description
`-cpu=<cpu>`	Select target CPU

The assembly file that RTA-OSEK generates from your OIL configuration file is called `osgen.tri`. 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_intvec_*`	ROM	RTA-OSEK interrupt wrappers
`os_pid`	ROM	RTA-OSEK read-only data
`os_pird`	ROM	RTA-OSEK initialization data
`os_pnird`	ROM	RTA-OSEK initialization data
`os_pir`	RAM	RTA-OSEK initialized data; initialized by StartOS()
`os_pir2`	RAM	RTA-OSEK initialized data; must be initialized during C-startup
`os_pnir`	RAM	RTA-OSEK SDA initialized data; initialized by StartOS()
`os_pnir2`	RAM	RTA-OSEK SDA initialized data; must be initialized during C-startup
`os_pur`	RAM	RTA-OSEK uninitialized data; must be zeroed during C-startup
`os_pnur`	RAM	RTA-OSEK SDA uninitialized data; must be zeroed during C-startup
`os_cntr`	RAM	RTA-OSEK SDA uninitialized data; must be zeroed during C-startup
`os_trace_ram`	RAM	RTA-TRACE uninitialized data; must be zeroed during C-startup

In some cases, sections produced by the linker must be located according to special constraints. The following table indicates which sections must be located with which particular constraints:

Sections	Constraints
`os_pnir`	Must be linked within 64KB after `.sdabase`
`os_pnir2`	Must be linked within 64KB after `.sdabase`
`os_pnur`	Must be linked within 64KB after `.sdabase`
`os_cntr`	Must be linked within 64KB after `.sdabase`

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.

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 (Build 13751-15499 or later)

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for TriCore/GreenHills. 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 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 Green Hills Software, Inc. C compiler can generate appropriate interrupt handling code for a C function decorated with the __interrupt function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

Category 2 ISRs are provided with a C function context by the RTA-OSEK Component, since the RTA-OSEK Component handles the interrupt context itself. The handlers are written using the OSEK OS standard ISR() macro, shown in Code Example 3:1.

```
#include “MyISR.h”
ISR(MyISR) {
  /* Handler routine */
}
```

Code Example 3:1 - Category 2 ISR Interrupt Handler

You must not insert a return from interrupt instruction in such a function. The return is handled automatically by the RTA-OSEK Component.

3.1.5 Vector Table Issues

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

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

The meaning of the option in the RTA-OSEK GUI to “generate vector table” is considered to be “use the Green Hills vector table in libarch.a.”

3.1.6 Vector Table Generation Enabled

If the option is enabled, Category 2 interrupt wrappers will be generated such that the Green Hills vector table jumps to them. Each wrapper will be placed in a unique section os_intvec_*vv*, where *vv* corresponds with the vector number. There is no restriction on where these sections may be placed, so, for example, they may be grouped together into one section in the linker directive file, as follows:

```
os_intvec ALIGN(2)   : {"*(os_intvec_*)"}   > .
```

Category 1 interrupt functions may be written using the `__interrupt` qualifier.

3.1.7 Vector Table Generation Disabled

If the “generate vector table” option is disabled, all references to the Green Hills vector table are removed. The interrupt wrappers should be placed directly in the vector table. Again, each Category 2 interrupt wrapper is generated in a unique section `os_intvec_`*`vv`*, which should be linked at an offset of (*vv* × 32) bytes from the base of the vector table. The BIV register must be initialized to point to the base of this custom vector table.

Category 1 interrupt functions should not be written using the `__interrupt` qualifier, as this will generate code to use the Green Hills vector table. A plain C function should be used instead, with inline assembler macros to correctly save and restore the lower context, and return from interrupt.

As an example, consider an application with a single Category 2 interrupt A, and a single Category 1 interrupt B.

ISR	Category	Vector	Priority
A	2	1	1
B	1	2	2

The Category 1 interrupt could be written as either:

```
#pragma ghs section text="vec_02"
void B(void)
{
  __bisr(2);
  …
  __rslcx();
  __asm("rfe16");
}
```

or:

```
#pragma ghs section text="vec_02"
void B(void)
{
  __svlcx();
  …
  __rslcx();
  __asm("rfe16");
}
```

The linker directive file could then be written as follows:

```
…
pad_vec_00    ALIGN(256)   PAD(32)   :   > .
os_intvec_01  ALIGN(32)              :   > .
vec_02        ALIGN(32)              :   > .
…
```

Then the BIV register could be initialized to the value of the linker-defined symbol __ghsbeginpad_vec_00.

3.1.8 IO Privilege Mode

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

3.1.9 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.

3.1.10 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

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

and

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

3.1.11 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.

3.1.12 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.

3.1.13 Default Interrupt

The default interrupt handler does not require the `__interrupt` 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 (initialised to the start of the `.interrupts` section by Green Hills C startup code)
FCX	Free context list. This should be initialized to point to a suitably large linked list of context areas (performed by Green Hills C startup code).
PSW.IO	Supervisor (0b10)

The RTA-OSEK Component uses the following hardware registers. They should 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.

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

3.3.2 Stack Usage within API Calls

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

Standard

API max usage (bytes): 0

Timing

API max usage (bytes): 0

Extended

API max usage (bytes): 8

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:

```
// Reserve space for the stack
.stack ALIGN(8) PAD(stack_reserve) :     > .
// Create the label for the initial SP value
_os_empty_stack_sp = .;
```

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	Internal scratchpad 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	276	276	280
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	348	348	352
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	84	84	84	84	84	84
	ROM	292	292	296	404	404	408
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	2	2	2	2	2	2
	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	80	80	80	80	80	80
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
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	84	84	84	84	84	84
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		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	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	114	114	114	114	114	114
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	122	122	122	122	122	122
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	80	80	80	80	80	80
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	84	84	84	84	84	84
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		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		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		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	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	126	126	126	126	126	126
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	134	134	134	134	134	134
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	84	84	84	84	84	84
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	84	84	84	84	84	84
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			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	168	210	254	176	220	294
	NS		134	192	236	142	202	276
	KL	2	80	128	172	88	138	208
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	140	196	240	148	206	268
	SWH	1, 9	178	244	288	186	254	322
	NSL	8	140	196	240	148	206	268
	NSH	9	168	234	278	176	244	312
Schedule			110	110	150	110	110	150
GetTaskID			28	28	28	28	28	28
GetTaskState			88	88	88	106	106	106
EnableAllInterrupts			30	30	30	30	30	30
DisableAllInterrupts			34	34	34	34	34	34
ResumeAllInterrupts			42	42	42	42	42	42
SuspendAllInterrupts			52	52	52	52	52	52
ResumeOSInterrupts			42	42	42	42	42	42
SuspendOSInterrupts			58	58	58	58	58	58
GetResource	Task	7	24	24	30	24	24	30
	Combined	6	80	80	80	80	80	80
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	82	82	82	82	82	82
	Combined	6	188	188	188	188	188	188
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	126	126	208
	NS		n/a	n/a	n/a	104	104	188
	NS1i	10	n/a	n/a	n/a	76	n/a	n/a
	KL	2	n/a	n/a	n/a	58	58	140
	KL1i	2, 10	n/a	n/a	n/a	24	n/a	n/a
ClearEvent			n/a	n/a	n/a	76	76	76

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	304	304	592
	fp	11	n/a	n/a	n/a	320	320	624
	1i	10	n/a	n/a	n/a	20	n/a	n/a
GetAlarmBase			72	72	72	72	72	72
GetAlarm			98	98	98	98	98	98
SetRelAlarm			572	572	572	572	572	572
SetAbsAlarm			546	546	546	546	546	546
CancelAlarm			88	88	88	88	88	88
InitCounter			68	68	68	68	68	68
GetCounterValue			82	82	82	82	82	82
GetScheduleTableStatus		34	54	74	74	54	74	74
NextScheduleTable		34	62	222	222	62	222	222
StartScheduleTable		34	108	166	166	108	166	166
StopScheduleTable		34	78	118	118	78	118	118
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		28	28	28	28	28	28
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	30	30	30	30	30	30
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		78	78	78	78	78	78
	KL	2	32	32	32	32	32	32
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			262	262	262	262	262	262
ShutdownOS	NoHook	12	42	42	42	42	42	42
	Hook	13	54	54	54	54	54	54
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			26	26	26	26	26	26
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	78	78	78	208	208	208

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	208	208	208	208	208	208
GetMessageResource			36	36	36	36	36	36
ReleaseMessageResource			34	34	34	34	34	34
GetMessageStatus			44	44	44	44	44	44
SendMessage	SW CCCA	1, 14	92	92	92	278	278	278
	SW CCCB	1, 15	264	264	264	278	278	278
	NS CCCA	14	92	92	92	278	278	278
	NS CCCB	15	264	264	264	278	278	278
	KL CCCA	2, 14	46	46	46	230	230	230
	KL CCCB	2, 15	216	216	216	230	230	230
main_dispatch	NoHook	12	166	166	204	166	166	204
	Hook	13	208	208	252	208	208	252
sub_dispatch	B1LF	19	22	22	22	22	22	22
	B1HI	20	102	102	102	102	102	102
	B1HF	21	110	110	110	110	110	110
	B2LI	22	n/a	84	106	n/a	84	106
	B2LF	23	n/a	90	112	n/a	90	112
	B2HI	24	n/a	304	364	n/a	304	364
	B2HF	25	n/a	314	372	n/a	314	372
	E1HI	26	n/a	n/a	n/a	492	492	590
	E1HF	27	n/a	n/a	n/a	502	502	576
	E2HI	28	n/a	n/a	n/a	n/a	n/a	590
	E2HF	29	n/a	n/a	n/a	n/a	n/a	576
ErrorHook support		16	40	40	40	40	40	40
	ServiceID	17	50	50	50	50	50	50
	Parameters	18	70	70	70	70	70	70
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	166	280	332	192	328	386
	NS		132	256	308	154	306	364
	KL	2	78	198	250	104	244	302
ChainTaskset	SWL	1, 8	156	282	318	164	292	358
	SWH	1, 9	196	330	380	204	338	420
	NSL	8	156	282	318	164	292	358
	NSH	9	186	320	370	194	328	410
GetTasksetRef			8	8	8	8	8	8

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
MergeTaskset			58	58	58	58	58	58
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			62	62	62	62	62	62
TestSubTaskset			62	62	62	62	62	62
TestEquivalentTaskset			60	60	60	60	60	60
TickSchedule	SW	1	242	182	182	182	182	182
	NS		204	154	154	154	154	154
	KL	2	156	106	106	106	106	106
AdvanceSchedule	SW	1	232	192	192	192	192	192
	NS		202	160	160	160	160	160
	KL	2	158	116	116	116	116	116
StartSchedule			84	84	84	84	84	84
StopSchedule			72	72	72	72	72	72
GetScheduleStatus			98	98	98	98	98	98
GetScheduleValue			82	82	82	82	82	82
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			42	42	42	42	42	42
GetExecutionTime			4	4	4	4	4	4
GetLargestExecutionTime			8	8	8	8	8	8
ResetLargestExecutionTime			4	4	4	4	4	4
GetStackOffset			20	20	20	20	20	20

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	168	210	254	176	220	294
	NS		134	192	236	142	202	276
	KL	2	80	128	172	88	138	208
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	140	196	240	148	206	268
	SWH	1, 9	178	244	288	186	254	322
	NSL	8	140	196	240	148	206	268
	NSH	9	168	234	278	176	244	312
Schedule			130	130	170	130	130	170
GetTaskID			28	28	28	28	28	28
GetTaskState			88	88	88	106	106	106
EnableAllInterrupts			30	30	30	30	30	30
DisableAllInterrupts			34	34	34	34	34	34
ResumeAllInterrupts			42	42	42	42	42	42
SuspendAllInterrupts			52	52	52	52	52	52
ResumeOSInterrupts			42	42	42	42	42	42
SuspendOSInterrupts			58	58	58	58	58	58
GetResource	Task	7	24	24	30	24	24	30
	Combined	6	80	80	80	80	80	80
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	106	106	106	106	106	106
	Combined	6	234	234	234	234	234	234
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	126	126	208
	NS		n/a	n/a	n/a	104	104	188
	NS1i	10	n/a	n/a	n/a	76	n/a	n/a
	KL	2	n/a	n/a	n/a	58	58	140
	KL1i	2, 10	n/a	n/a	n/a	24	n/a	n/a
ClearEvent			n/a	n/a	n/a	76	76	76
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	406	406	694
	fp	11	n/a	n/a	n/a	422	422	726

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	116	n/a	n/a
GetAlarmBase			72	72	72	72	72	72
GetAlarm			98	98	98	98	98	98
SetRelAlarm			572	572	572	572	572	572
SetAbsAlarm			546	546	546	546	546	546
CancelAlarm			88	88	88	88	88	88
InitCounter			68	68	68	68	68	68
GetCounterValue			82	82	82	82	82	82
GetScheduleTableStatus		34	54	74	74	54	74	74
NextScheduleTable		34	62	222	222	62	222	222
StartScheduleTable		34	108	166	166	108	166	166
StopScheduleTable		34	78	118	118	78	118	118
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		28	28	28	28	28	28
GetISRID		4	34	34	34	34	34	34
Process container	Yielding	32	30	30	30	30	30	30
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		78	78	78	78	78	78
	KL	2	32	32	32	32	32	32
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			334	334	334	334	334	334
ShutdownOS	NoHook	12	42	42	42	42	42	42
	Hook	13	54	54	54	54	54	54
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			26	26	26	26	26	26
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	78	78	78	208	208	208
	CCCB	15	208	208	208	208	208	208
GetMessageResource			36	36	36	36	36	36
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			44	44	44	44	44	44
SendMessage	SW CCCA	1, 14	92	92	92	278	278	278
	SW CCCB	1, 15	264	264	264	278	278	278
	NS CCCA	14	92	92	92	278	278	278
	NS CCCB	15	264	264	264	278	278	278
	KL CCCA	2, 14	46	46	46	230	230	230
	KL CCCB	2, 15	216	216	216	230	230	230
main_dispatch	NoHook	12	204	204	246	204	204	246
	Hook	13	242	242	294	242	242	294
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	94	94	94	94	94	94
	B1HF	21	102	102	102	102	102	102
	B2LI	22	n/a	48	70	n/a	48	70
	B2LF	23	n/a	54	76	n/a	54	76
	B2HI	24	n/a	238	294	n/a	238	294
	B2HF	25	n/a	246	300	n/a	246	300
	E1HI	26	n/a	n/a	n/a	478	478	572
	E1HF	27	n/a	n/a	n/a	486	486	552
	E2HI	28	n/a	n/a	n/a	n/a	n/a	572
	E2HF	29	n/a	n/a	n/a	n/a	n/a	552
ErrorHook support		16	40	40	40	40	40	40
	ServiceID	17	50	50	50	50	50	50
	Parameters	18	70	70	70	70	70	70
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	94	94	94	94	94	94
Timing_termination		4	88	88	88	88	88	88
ActivateTaskset	SW	1	166	280	332	192	328	386
	NS		132	256	308	154	306	364
	KL	2	78	198	250	104	244	302
ChainTaskset	SWL	1, 8	156	282	318	164	292	358
	SWH	1, 9	196	330	380	204	338	420
	NSL	8	156	282	318	164	292	358
	NSH	9	186	320	370	194	328	410
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			58	58	58	58	58	58
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			62	62	62	62	62	62

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestSubTaskset			62	62	62	62	62	62
TestEquivalentTaskset			60	60	60	60	60	60
TickSchedule	SW	1	242	182	182	182	182	182
	NS		204	154	154	154	154	154
	KL	2	156	106	106	106	106	106
AdvanceSchedule	SW	1	232	192	192	192	192	192
	NS		202	160	160	160	160	160
	KL	2	158	116	116	116	116	116
StartSchedule			84	84	84	84	84	84
StopSchedule			72	72	72	72	72	72
GetScheduleStatus			98	98	98	98	98	98
GetScheduleValue			82	82	82	82	82	82
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			42	42	42	42	42	42
GetExecutionTime			128	128	128	128	128	128
GetLargestExecutionTime			16	16	16	16	16	16
ResetLargestExecutionTime			14	14	14	14	14	14
GetStackOffset			20	20	20	20	20	20

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	278	328	370	298	342	400
	NS		330	396	436	356	410	488
	KL	2	154	202	244	174	216	274
TerminateTask	LExt	3	166	166	166	166	166	166

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	132	132	132	132	132	132
ChainTask	SWL	1, 8	306	352	394	320	364	430
	SWH	1, 9	334	384	428	348	396	456
	NSL	8	370	414	456	384	426	492
	NSH	9	372	422	466	386	434	494
Schedule			276	276	320	276	276	320
GetTaskID			48	48	48	48	48	48
GetTaskState			298	298	298	282	282	282
EnableAllInterrupts			52	52	52	52	52	52
DisableAllInterrupts			56	56	56	56	56	56
ResumeAllInterrupts			102	102	102	102	102	102
SuspendAllInterrupts			72	72	72	72	72	72
ResumeOSInterrupts			100	100	100	100	100	100
SuspendOSInterrupts			78	78	78	78	78	78
GetResource	Task	7	356	356	322	356	356	322
	Combined	6	376	376	376	376	376	376
	CLEx	3	366	366	366	366	366	366
ReleaseResource	Task	7	346	346	346	346	346	346
	Combined	6	480	480	480	480	480	480
	CLEx	3	302	302	302	302	302	302
SetEvent	SW	1	n/a	n/a	n/a	332	332	414
	NS		n/a	n/a	n/a	364	364	450
	NS1i	10	n/a	n/a	n/a	266	n/a	n/a
	KL	2	n/a	n/a	n/a	214	214	294
	KL1i	2, 10	n/a	n/a	n/a	156	n/a	n/a
ClearEvent			n/a	n/a	n/a	148	148	148
GetEvent			n/a	n/a	n/a	154	154	154
WaitEvent	<default>		n/a	n/a	n/a	526	526	810
	fp	11	n/a	n/a	n/a	542	542	842
	1i	10	n/a	n/a	n/a	224	n/a	n/a
GetAlarmBase			212	212	212	212	212	212
GetAlarm			214	214	214	214	214	214
SetRelAlarm			704	704	704	704	704	704
SetAbsAlarm			708	708	708	708	708	708
CancelAlarm			200	200	200	200	200	200
InitCounter			288	288	288	288	288	288
GetCounterValue			232	232	232	232	232	232

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	72	92	92	72	92	92
NextScheduleTable		34	82	292	292	82	292	292
StartScheduleTable		34	138	206	206	138	206	206
StopScheduleTable		34	108	158	158	108	158	158
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		30	30	30	30	30	30
GetISRID		4	54	54	54	54	54	54
Process container	Yielding	32	30	30	30	30	30	30
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		132	132	132	132	132	132
	KL	2	32	32	32	32	32	32
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			356	356	356	356	356	356
ShutdownOS	NoHook	12	52	52	52	52	52	52
	Hook	13	64	64	64	64	64	64
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			46	46	46	46	46	46
StopCOM			60	60	60	60	60	60
ReadFlag			32	32	32	32	32	32
ResetFlag			36	36	36	36	36	36
ReceiveMessage	CCCA	14	186	186	186	314	314	314
	CCCB	15	314	314	314	314	314	314
GetMessageResource			84	84	84	84	84	84
ReleaseMessageResource			84	84	84	84	84	84
GetMessageStatus			118	118	118	118	118	118
SendMessage	SW CCCA	1, 14	244	244	244	422	422	422
	SW CCCB	1, 15	408	408	408	422	422	422
	NS CCCA	14	244	244	244	422	422	422
	NS CCCB	15	408	408	408	422	422	422
	KL CCCA	2, 14	162	162	162	340	340	340
	KL CCCB	2, 15	326	326	326	340	340	340
main_dispatch	NoHook	12	204	204	246	204	204	246

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	242	242	294	242	242	294
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	94	94	94	94	94	94
	B1HF	21	102	102	102	102	102	102
	B2LI	22	n/a	48	70	n/a	48	70
	B2LF	23	n/a	54	76	n/a	54	76
	B2HI	24	n/a	238	294	n/a	238	294
	B2HF	25	n/a	246	300	n/a	246	300
	E1HI	26	n/a	n/a	n/a	478	478	572
	E1HF	27	n/a	n/a	n/a	486	486	552
	E2HI	28	n/a	n/a	n/a	n/a	n/a	572
	E2HF	29	n/a	n/a	n/a	n/a	n/a	552
ErrorHook support		16	148	148	148	148	148	148
	ServiceID	17	158	158	158	158	158	158
	Parameters	18	178	178	178	178	178	178
validity_checks		3	28	28	28	28	28	28
Timing_dispatch		4	94	94	94	94	94	94
Timing_termination		4	88	88	88	88	88	88
ActivateTaskset	SW	1	332	404	448	372	432	490
	NS		366	438	482	408	468	526
	KL	2	206	288	334	256	322	386
ChainTaskset	SWL	1, 8	374	442	486	408	460	522
	SWH	1, 9	418	490	534	452	510	570
	NSL	8	412	480	524	446	498	560
	NSH	9	446	518	562	480	538	598
GetTasksetRef			122	122	122	122	122	122
MergeTaskset			314	314	314	314	314	314
AssignTaskset			186	186	186	186	186	186
RemoveTaskset			318	318	318	318	318	318
TestSubTaskset			342	342	342	342	342	342
TestEquivalentTaskset			340	340	340	340	340	340
TickSchedule	SW	1	384	344	344	344	344	344
	NS		446	440	440	440	440	440
	KL	2	276	232	232	232	232	232
AdvanceSchedule	SW	1	374	338	338	338	338	338
	NS		436	454	454	454	454	454
	KL	2	300	262	262	262	262	262

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			274	274	274	274	274	274
StopSchedule			230	230	230	230	230	230
GetScheduleStatus			262	262	262	262	262	262
GetScheduleValue			220	220	220	220	220	220
GetScheduleNext			94	94	94	94	94	94
SetScheduleNext			184	184	184	184	184	184
GetArrivalpointDelay			132	132	132	132	132	132
SetArrivalpointDelay			154	154	154	154	154	154
GetArrivalpointTasksetRef			130	130	130	130	130	130
GetArrivalpointNext			132	132	132	132	132	132
SetArrivalpointNext			198	198	198	198	198	198
TestArrivalpointWritable			158	158	158	158	158	158
GetExecutionTime			196	196	196	196	196	196
GetLargestExecutionTime			104	104	104	104	104	104
ResetLargestExecutionTime			98	98	98	98	98	98
GetStackOffset			20	20	20	20	20	20

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	42	57	67	44	53	75
	NS	33	49	58	35	46	66
	KL	22	37	46	25	34	57
TerminateTask	LExt	0	0	0	0	0	0
	H	98	97	97	97	98	96
ChainTask	SWL	119	144	171	150	159	199
	SWH	180	201	230	211	218	258
	NSL	119	143	170	149	158	198
	NSH	178	199	227	208	218	255
Schedule	SW	33	36	39	34	34	39
GetTaskID		18	18	19	18	18	18
GetTaskState		35	35	35	42	41	40
EnableAllInterrupts		21	21	22	21	22	21
DisableAllInterrupts		16	16	16	17	16	16
ResumeAllInterrupts		25	25	25	25	25	25
SuspendAllInterrupts		18	18	17	18	17	18
ResumeOSInterrupts		25	25	25	25	25	25
SuspendOSInterrupts		18	18	17	18	17	18
GetResource	Task	16	16	19	16	16	18
	Combined	27	27	28	27	27	27
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	35	33	33	33	33	33
	Combined	49	49	50	49	51	49
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	40	41	40
	NS	n/a	n/a	n/a	39	39	39
	KL	n/a	n/a	n/a	26	26	26
ClearEvent		n/a	n/a	n/a	27	27	27
GetEvent		n/a	n/a	n/a	12	12	12
WaitEvent	<default>	n/a	n/a	n/a	207	209	223
	fp	n/a	n/a	n/a	211	212	224
GetAlarmBase		30	30	30	30	30	30
GetAlarm		33	33	33	33	32	33

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		47	48	47	47	47	48
SetAbsAlarm		45	45	46	44	45	45
CancelAlarm		28	28	28	28	28	28
InitCounter		30	30	28	28	28	28
GetCounterValue		30	30	30	30	30	30
osek_tick_alarm	<default>	34	33	34	34	33	33
	KL	19	19	19	19	19	19
osek_incr_counter		5	5	5	5	5	5
GetActiveApplicationMode		4	4	4	4	4	4
StartOS		1669	1668	1670	1669	1669	1670
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	19	20	19	19	19	19
InitCOM		6	6	6	7	7	7
CloseCOM		6	6	6	7	7	7
StartCOM		25	25	26	84	85	84
StopCOM		10	10	10	11	11	11
ReadFlag		n/a	n/a	n/a	9	9	9
ResetFlag		n/a	n/a	n/a	7	7	7
ReceiveMessage		24	24	24	102	102	102
GetMessageResource		n/a	n/a	n/a	40	40	39
ReleaseMessageResource		n/a	n/a	n/a	55	55	55
GetMessageStatus		n/a	n/a	n/a	25	26	25
SendMessage	SW	79	94	104	178	187	210
	NS	71	86	95	170	182	200
	KL	46	60	69	144	152	176
ActivateTaskset	SW	40	245	294	42	277	270
	NS	33	266	283	36	272	260
	KL	19	255	241	20	224	277
	SW2	40	245	294	42	277	270
	NS2	33	266	283	36	272	260
	KL2	19	255	241	20	224	277
ChainTaskset	SWL	116	355	393	144	379	388
	SWH	178	388	454	206	440	479
	NSL	116	323	394	144	378	388
	NSH	176	416	453	204	437	476
GetTasksetRef		9	9	9	9	9	9

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
MergeTaskset		24	24	24	26	25	25
AssignTaskset		9	9	9	9	9	9
RemoveTaskset		24	24	24	24	25	24
TestSubTaskset		24	24	24	25	24	24
TestEquivalentTaskset		25	26	25	26	25	25
TickSchedule	SW	64	318	304	84	293	346
	NS	55	307	293	72	283	335
	KL	41	295	279	60	267	322
	SW2	64	318	304	84	289	341
	NS2	55	307	293	72	278	329
	KL2	41	295	279	60	264	317
AdvanceSchedule	SW	57	316	300	80	288	344
	NS	49	304	289	69	277	332
	KL	35	289	275	55	263	317
	SW2	57	316	300	80	284	339
	NS2	49	304	289	69	273	327
	KL2	35	289	275	55	259	312
StartSchedule		40	40	40	39	39	39
StopSchedule		36	37	36	37	37	37
GetScheduleStatus		38	38	38	38	38	38
GetScheduleValue		36	36	37	36	37	36
GetScheduleNext		10	10	10	10	10	10
SetScheduleNext		11	11	11	10	10	10
GetArrivalpointDelay		9	9	9	9	9	9
SetArrivalpointDelay		8	8	8	8	8	8
GetArrivalpointTasksetRef		8	8	8	8	8	8
GetArrivalpointNext		9	9	9	9	9	9
SetArrivalpointNext		8	8	8	8	8	8
TestArrivalpointWritable		20	20	20	20	20	20
GetExecutionTime		6	6	6	6	6	6
GetLargestExecutionTime		11	11	11	11	11	11
ResetLargestExecutionTime		7	7	7	7	7	7
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	42	59	67	44	53	74
	NS	33	49	59	36	45	66
	KL	22	37	47	25	34	55
TerminateTask	LExt	0	0	0	0	0	0
	H	198	197	199	199	198	199
ChainTask	SWL	218	244	272	251	264	304
	SWH	288	313	339	322	331	371
	NSL	218	244	274	252	264	304
	NSH	286	309	337	319	329	367
Schedule	SW	33	35	38	34	34	39
GetTaskID		17	18	18	18	18	19
GetTaskState		35	35	37	40	41	40
EnableAllInterrupts		21	21	21	21	21	21
DisableAllInterrupts		17	16	15	15	15	16
ResumeAllInterrupts		9	9	9	9	25	9
SuspendAllInterrupts		11	11	12	11	18	11
ResumeOSInterrupts		9	9	9	9	25	9
SuspendOSInterrupts		11	11	12	11	18	11
GetResource	Task	16	16	18	16	16	18
	Combined	27	27	27	27	27	27
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	30	30	30	31	31	31
	Combined	50	50	49	49	49	51
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	40	40	40
	NS	n/a	n/a	n/a	39	39	39
	KL	n/a	n/a	n/a	26	26	26
ClearEvent		n/a	n/a	n/a	27	27	27
GetEvent		n/a	n/a	n/a	12	12	12
WaitEvent	<default>	n/a	n/a	n/a	295	294	314
	fp	n/a	n/a	n/a	299	299	318
GetAlarmBase		30	30	30	30	30	30
GetAlarm		33	33	33	32	33	33

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		47	47	47	48	48	47
SetAbsAlarm		44	44	45	45	45	44
CancelAlarm		28	28	28	27	28	28
InitCounter		29	29	29	28	28	28
GetCounterValue		30	33	30	30	30	30
osek_tick_alarm	<default>	34	34	34	34	34	34
	KL	19	19	19	19	19	19
osek_incr_counter		5	5	5	5	5	5
GetActiveApplicationMode		4	4	4	4	4	4
StartOS		5174	5195	5172	5172	5171	5172
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	19	20	19	19	19	19
InitCOM		6	6	6	7	7	7
CloseCOM		6	6	6	7	7	7
StartCOM		25	25	26	88	88	88
StopCOM		10	10	10	11	11	11
ReadFlag		n/a	n/a	n/a	9	9	9
ResetFlag		n/a	n/a	n/a	7	7	7
ReceiveMessage		24	23	24	101	101	102
GetMessageResource		n/a	n/a	n/a	39	39	39
ReleaseMessageResource		n/a	n/a	n/a	52	52	55
GetMessageStatus		n/a	n/a	n/a	25	25	25
SendMessage	SW	79	97	104	178	187	208
	NS	70	86	97	170	178	201
	KL	47	60	70	144	154	174
ActivateTaskset	SW	40	276	292	41	278	298
	NS	33	267	252	36	301	291
	KL	19	286	303	20	256	277
	SW2	40	276	292	41	278	298
	NS2	33	267	252	36	301	291
	KL2	19	286	303	20	256	277
ChainTaskset	SWL	215	458	527	247	482	520
	SWH	291	497	534	317	552	621
	NSL	215	457	500	246	484	521
	NSH	286	526	562	315	549	560
GetTasksetRef		9	9	9	9	9	9

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		24	24	24	25	25	25
AssignTaskset		9	9	9	9	9	9
RemoveTaskset		24	24	25	24	24	24
TestSubTaskset		24	24	24	24	24	24
TestEquivalentTaskset		26	25	25	25	25	25
TickSchedule	SW	64	348	366	83	325	348
	NS	55	338	355	72	311	337
	KL	41	325	344	61	300	322
	SW2	64	348	366	83	319	342
	NS2	55	338	355	72	308	332
	KL2	41	325	343	61	297	317
AdvanceSchedule	SW	58	345	362	81	320	342
	NS	49	334	351	70	308	331
	KL	35	320	338	55	294	317
	SW2	58	345	362	81	316	337
	NS2	49	334	351	70	305	326
	KL2	35	320	339	55	291	312
StartSchedule		39	40	40	39	39	39
StopSchedule		36	36	37	36	36	36
GetScheduleStatus		38	38	38	38	38	39
GetScheduleValue		36	38	36	36	36	36
GetScheduleNext		10	10	10	10	10	10
SetScheduleNext		11	11	11	10	10	10
GetArrivalpointDelay		9	9	9	9	9	9
SetArrivalpointDelay		8	8	8	8	8	8
GetArrivalpointTasksetRef		8	8	8	8	8	8
GetArrivalpointNext		9	9	9	9	9	9
SetArrivalpointNext		8	8	8	8	8	8
TestArrivalpointWritable		20	20	20	20	20	20
GetExecutionTime		52	52	54	52	52	54
GetLargestExecutionTime		13	13	13	13	13	13
ResetLargestExecutionTime		10	10	10	10	10	10
GetStackOffset		10	10	10	10	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	160	179	191	166	174	194
	NS	172	195	206	179	190	211
	KL	134	150	162	135	146	165
TerminateTask	LExt	180	181	179	178	179	179
	H	241	241	240	238	238	241
ChainTask	SWL	368	395	423	403	412	449
	SWH	435	458	488	466	475	515
	NSL	385	411	438	422	427	467
	NSH	453	476	507	485	493	532
Schedule	SW	59	59	61	59	60	60
GetTaskID		22	21	21	21	21	21
GetTaskState		166	168	168	168	168	168
EnableAllInterrupts		26	26	27	26	26	26
DisableAllInterrupts		22	20	20	20	20	20
ResumeAllInterrupts		17	18	17	19	31	19
SuspendAllInterrupts		15	15	16	15	22	15
ResumeOSInterrupts		17	18	17	19	31	19
SuspendOSInterrupts		15	15	16	15	22	15
GetResource	Task	228	227	120	248	248	140
	Combined	119	117	120	137	138	139
	CLEx	128	127	128	148	149	148
ReleaseResource	Task	119	121	120	140	142	141
	Combined	125	126	125	146	147	147
	CLEx	115	115	114	135	135	135
SetEvent	SW	n/a	n/a	n/a	171	170	170
	NS	n/a	n/a	n/a	182	182	179
	KL	n/a	n/a	n/a	148	148	149
ClearEvent		n/a	n/a	n/a	44	42	42
GetEvent		n/a	n/a	n/a	134	134	134
WaitEvent	<default>	n/a	n/a	n/a	338	337	348
	fp	n/a	n/a	n/a	340	341	351
GetAlarmBase		105	105	112	104	104	113
GetAlarm		105	106	111	104	104	112

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		129	129	136	128	128	136
SetAbsAlarm		125	125	129	121	121	132
CancelAlarm		99	98	106	99	99	106
InitCounter		166	164	175	166	165	177
GetCounterValue		100	99	98	98	99	98
osek_tick_alarm	<default>	44	44	44	44	44	44
	KL	18	18	18	18	18	18
osek_incr_counter		4	4	4	4	4	4
GetActiveApplicationMode		4	4	4	4	4	4
StartOS		5353	5354	5353	5353	5352	5354
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	21	21	22	23	21	21
InitCOM		7	7	7	7	7	7
CloseCOM		7	7	7	7	7	7
StartCOM		32	32	32	92	91	93
StopCOM		17	19	17	17	17	18
ReadFlag		n/a	n/a	n/a	18	18	18
ResetFlag		n/a	n/a	n/a	15	15	15
ReceiveMessage		78	81	79	157	157	155
GetMessageResource		n/a	n/a	n/a	205	206	207
ReleaseMessageResource		n/a	n/a	n/a	199	200	200
GetMessageStatus		n/a	n/a	n/a	66	66	66
SendMessage	SW	255	274	286	358	366	384
	NS	268	292	303	369	380	402
	KL	206	222	233	303	314	333
ActivateTaskset	SW	307	550	565	347	549	606
	NS	319	622	577	330	531	555
	KL	280	489	507	289	524	545
	SW2	307	550	565	347	549	606
	NS2	319	622	577	330	531	555
	KL2	280	489	507	289	523	545
ChainTaskset	SWL	519	738	801	561	757	866
	SWH	563	871	875	629	830	872
	NSL	534	755	816	574	743	850
	NSH	573	833	889	642	874	949
GetTasksetRef		124	123	123	123	123	123

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		83	83	83	84	83	83
AssignTaskset		55	55	55	55	55	55
RemoveTaskset		84	82	82	82	83	83
TestSubTaskset		90	89	89	88	88	88
TestEquivalentTaskset		87	87	87	87	87	87
TickSchedule	SW	96	576	593	376	616	640
	NS	103	587	603	384	628	652
	KL	67	545	564	343	586	608
	SW2	96	577	593	376	609	632
	NS2	103	587	603	384	622	644
	KL2	63	545	563	343	579	600
AdvanceSchedule	SW	93	573	593	373	615	638
	NS	104	583	600	384	624	649
	KL	61	542	561	340	582	606
	SW2	93	573	591	373	608	630
	NS2	104	583	600	382	617	641
	KL2	60	542	561	340	575	598
StartSchedule		79	77	77	78	77	77
StopSchedule		64	64	65	63	63	64
GetScheduleStatus		65	65	66	66	65	65
GetScheduleValue		63	63	64	63	63	63
GetScheduleNext		27	26	26	25	25	25
SetScheduleNext		52	51	52	52	51	53
GetArrivalpointDelay		34	34	34	33	34	36
SetArrivalpointDelay		43	43	43	43	43	44
GetArrivalpointTasksetRef		27	26	26	26	26	26
GetArrivalpointNext		27	28	28	28	28	29
SetArrivalpointNext		57	57	58	57	57	57
TestArrivalpointWritable		36	35	36	36	35	36
GetExecutionTime		65	66	65	65	65	65
GetLargestExecutionTime		118	120	119	119	119	121
ResetLargestExecutionTime		116	116	115	117	115	116
GetStackOffset		10	10	10	10	10	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	19	19	19	19	19	19
	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	19	19	19	19	19	19
	Cat 2	102	112	111	114	112	112

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	19	19	19	19	19	19
	Cat 2	103	113	113	113	115	115

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.

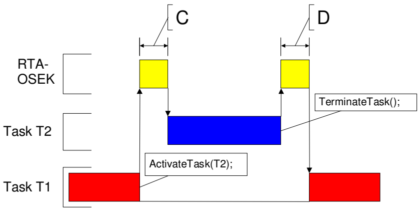

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

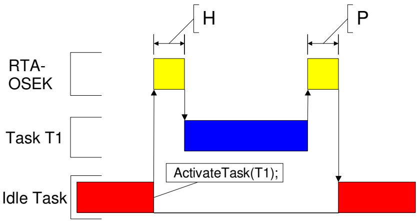

Figure 3: Task Activation from Idle Task

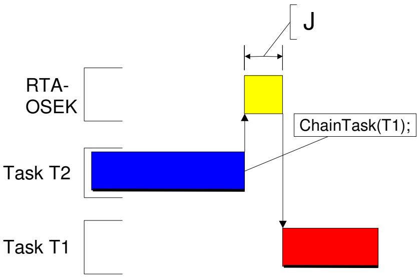

Figure 2: Task Chaining

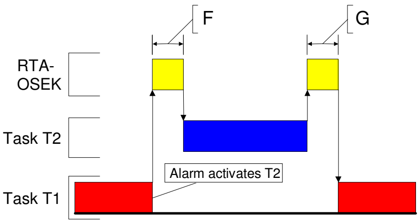

Figure 4: Task Activation from an Alarm

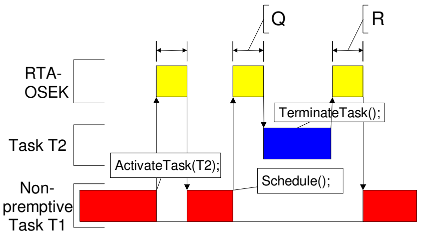

Figure 5: Non-Premptive Task Calls Schedule()

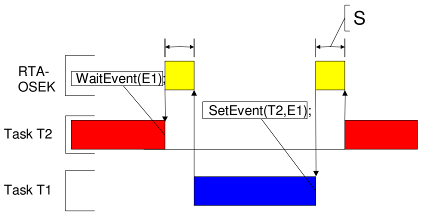

Figure 7: Waiting Task Activated by SetEvent()

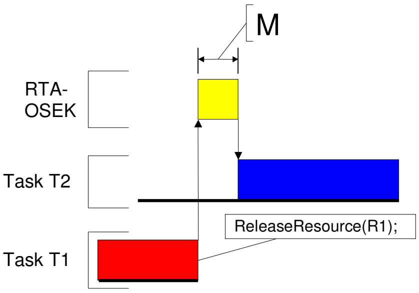

Figure 6: Blocked Task Activated by ReleaseResource()

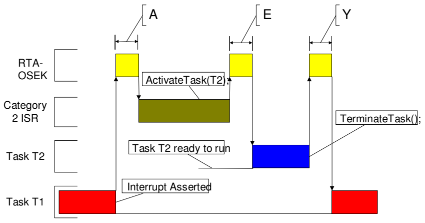

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	41	61	69	40	61	69
Figure 1: D	Heavy, Basic/Extended	82	92	103	101	102	106
ChainTask	Light, Basic	77	102	128	76	101	139
Figure 2: J	Heavy, Basic/Extended	241	274	312	261	283	326
Pre-emption	Light, Basic	66	92	121	69	92	133
Figure 1: C	Heavy, Basic/Extended	110	133	163	143	151	194
From idle task	Light, Basic	66	92	121	68	91	132
Figure 3: H	Heavy, Basic/Extended	110	133	163	142	150	193
Triggered by alarm	Light, Basic	108	132	162	110	132	173
Figure 4: F	Heavy, Basic/Extended	155	176	207	186	193	236
Schedule	Light, Basic	58	70	93	58	68	92
Figure 5: Q	Heavy, Basic/Extended	102	111	135	132	131	156
Release resource	Light, Basic	64	75	95	64	75	94
Figure 6: M	Heavy, Basic/Extended	109	116	137	138	138	158
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	196	196	246
From category 2 ISR	Light, Basic	62	83	103	73	82	102
Figure 8: E	Heavy, Basic/Extended	106	125	146	147	146	167

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	131	145	156	131	145	157
Figure 1: D	Heavy, Basic/Extended	179	189	201	200	200	210
ChainTask	Light, Basic	176	201	230	178	201	243
Figure 2: J	Heavy, Basic/Extended	446	480	518	472	491	541
Pre-emption	Light, Basic	133	158	185	137	155	196
Figure 1: C	Heavy, Basic/Extended	175	201	228	210	220	258
From idle task	Light, Basic	133	158	185	136	154	195
Figure 3: H	Heavy, Basic/Extended	175	201	228	209	217	257
Triggered by alarm	Light, Basic	175	200	227	179	196	237
Figure 4: F	Heavy, Basic/Extended	220	245	272	253	261	301
Schedule	Light, Basic	125	133	156	126	131	156
Figure 5: Q	Heavy, Basic/Extended	167	176	199	199	198	222
Release resource	Light, Basic	130	137	156	130	135	156
Figure 6: M	Heavy, Basic/Extended	172	180	199	203	202	222
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	246	245	293
From category 2 ISR	Light, Basic	221	240	261	234	239	262
Figure 8: E	Heavy, Basic/Extended	261	282	302	305	304	326

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	178	193	199	178	193	201
Figure 1: D	Heavy, Basic/Extended	223	232	242	240	240	249
ChainTask	Light, Basic	326	352	379	330	351	388
Figure 2: J	Heavy, Basic/Extended	635	669	709	654	673	723
Pre-emption	Light, Basic	247	270	303	251	270	310
Figure 1: C	Heavy, Basic/Extended	289	313	347	325	332	372
From idle task	Light, Basic	247	270	303	251	270	310
Figure 3: H	Heavy, Basic/Extended	289	313	347	328	332	372
Triggered by alarm	Light, Basic	302	325	357	306	325	365
Figure 4: F	Heavy, Basic/Extended	346	371	403	381	389	429
Schedule	Light, Basic	149	153	176	147	154	176
Figure 5: Q	Heavy, Basic/Extended	190	196	220	220	220	242
Release resource	Light, Basic	207	214	232	226	234	255
Figure 6: M	Heavy, Basic/Extended	249	255	276	300	300	322
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	370	371	419
From category 2 ISR	Light, Basic	239	257	275	250	255	275
Figure 8: E	Heavy, Basic/Extended	279	299	317	321	319	340

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		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	12	13	n/a	12	13
BCC2 heavyweight, floating-point		n/a	12	13	n/a	12	13
ECC1 heavyweight, integer		n/a	n/a	n/a	14	14	15
ECC1 heavyweight, floating-point		n/a	n/a	n/a	14	14	15
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	15
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	15
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	13	13	n/a	13	13
BCC2 heavyweight, floating-point		n/a	13	13	n/a	13	13
ECC1 heavyweight, integer		n/a	n/a	n/a	15	15	15
ECC1 heavyweight, floating-point		n/a	n/a	n/a	15	15	15
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	15
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	15

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	14	14	15	14	14	15
BCC1 lightweight, floating-point	15	15	16	15	15	16
BCC1 heavyweight, integer	32	32	33	32	32	33
BCC1 heavyweight, floating-point	32	32	33	32	32	33
BCC2 lightweight, integer	n/a	15	16	n/a	15	16
BCC2 lightweight, floating-point	n/a	15	16	n/a	15	16
BCC2 heavyweight, integer	n/a	16	17	n/a	16	17
BCC2 heavyweight, floating-point	n/a	16	17	n/a	16	17
ECC1 heavyweight, integer	n/a	n/a	n/a	17	17	18
ECC1 heavyweight, floating-point	n/a	n/a	n/a	17	17	18
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	18
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	18
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	15	15	15	15	15	15
BCC1 lightweight, floating-point	16	16	16	16	16	16
BCC1 heavyweight, integer	33	33	33	33	33	33
BCC1 heavyweight, floating-point	33	33	33	33	33	33
BCC2 lightweight, integer	n/a	16	16	n/a	16	16
BCC2 lightweight, floating-point	n/a	16	16	n/a	16	16
BCC2 heavyweight, integer	n/a	17	17	n/a	17	17
BCC2 heavyweight, floating-point	n/a	17	17	n/a	17	17
ECC1 heavyweight, integer	n/a	n/a	n/a	18	18	18
ECC1 heavyweight, floating-point	n/a	n/a	n/a	18	18	18
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	18
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	18

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	14	14	15	14	14	15
BCC1 lightweight, floating-point	15	15	16	15	15	16
BCC1 heavyweight, integer	32	32	33	32	32	33
BCC1 heavyweight, floating-point	32	32	33	32	32	33
BCC2 lightweight, integer	n/a	15	16	n/a	15	16
BCC2 lightweight, floating-point	n/a	15	16	n/a	15	16
BCC2 heavyweight, integer	n/a	16	17	n/a	16	17
BCC2 heavyweight, floating-point	n/a	16	17	n/a	16	17
ECC1 heavyweight, integer	n/a	n/a	n/a	17	17	18
ECC1 heavyweight, floating-point	n/a	n/a	n/a	17	17	18
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	18
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	18
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	15	15	15	15	15	15
BCC1 lightweight, floating-point	16	16	16	16	16	16
BCC1 heavyweight, integer	33	33	33	33	33	33
BCC1 heavyweight, floating-point	33	33	33	33	33	33
BCC2 lightweight, integer	n/a	16	16	n/a	16	16
BCC2 lightweight, floating-point	n/a	16	16	n/a	16	16
BCC2 heavyweight, integer	n/a	17	17	n/a	17	17
BCC2 heavyweight, floating-point	n/a	17	17	n/a	17	17
ECC1 heavyweight, integer	n/a	n/a	n/a	18	18	18
ECC1 heavyweight, floating-point	n/a	n/a	n/a	18	18	18
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	18
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	18

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for TriCore/GreenHills 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()`
`ResumeAllInterrupts()`	`osResumeAllInterrupts()`

6 Version 5.0.1

6.1 Workaround for Silicon Issue

Version 5.0.1 has a workaround for a race condition concerning the MTCR instruction and the contents of the PCXI register.

Version 5.0.1 libraries cannot be used with applications configured for 5.0.0; and version 5.0.0 libraries cannot be used with applications configured for 5.0.1. This avoids a problem that it was possible to link 5.0.1 libraries with applications configured for 5.0.0, giving an invalid configuration.

6.2 Section Names

The `os_cntr_bss` section has been renamed to `os_cntr` but still should reside in the SDA (.sbss) area.

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.