RTA-OSEK

Binding Manual: TriCore17x6/Tasking

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

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

www.etas.de

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

Japan

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

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

www.etas.co.jp

France

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

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

www.etas.fr

Korea

ETAS Korea Co. Ltd.
4F, 705 Bldg. 70-5
Yangjae-dong, Seocho-gu
Seoul 137-889, Korea

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

www.etas.co.kr

Great Britain

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

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

www.etas-uk.net

Copyright Notice

Version: RM00071-004

Disclaimer

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

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

Trademarks

RTA-OSEK and LiveDevices are trademarks of LiveDevices Ltd.

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

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

Contents

1 About this Guide

This guide provides port specific information for the TriCore17x6/Tasking implementation of LiveDevices' RTA-OSEK.

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

1.1 Who Should Read this Guide?

It is assumed that you are a developer. You should read this guide if you want to know low-level technical information to integrate the RTA-OSEK Component into your application.

1.2 Conventions

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

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

In this guide you'll see that program code, header file names, C type names, C functions and RTA-OSEK API call names all appear in the `courier` typeface. When the name of an object is made available to the programmer the name also appears in the `courier` typeface, so, for example, a task named Task1 appears as a task handle called `Task1`.

2 Toolchain Issues

In this chapter, you'll see the important details that you need to know about RTA-OSEK and your toolchain. A port of the RTA-OSEK Component is specific to both the target hardware *and* the compiler toolchain. You must make sure that you build your application with this toolchain.

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

2.1 Compiler

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

Vendor	Tasking
Compiler	TriCore VX-toolset C compiler
Version	v2.2r3 patch 1 Build 156.1.4

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

Option	Description
`-C<cpu>`	Selects target CPU
`--silicon-bug=cpu-tc013, cpu-tc048, cpu-tc060, cpu-tc065, cpu-tc068, cpu-tc069, cpu-tc070, cpu-tc071, cpu-tc072, cpu-tc074, cpu-tc081, cpu-tc082, cpu-tc083, cpu-tc094, cpu-tc095, cpu-tc096`	Includes workarounds for Tc17X6 CPU silicon bugs

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

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

Option	Description
`-C<cpu>`	Selects target CPU
`--silicon-bug=cpu-tc013, cpu-tc048, cpu-tc060, cpu-tc065, cpu-tc068, cpu-tc069, cpu-tc070, cpu-tc071, cpu-tc072, cpu-tc074, cpu-tc081, cpu-tc082, cpu-tc083, cpu-tc094, cpu-tc095, cpu-tc096`	Includes workarounds for Tc17X6 CPU silicon bugs
`--integer-enumeration`	Forces C enums to be integers

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

Option	Description
`-g`	Enable 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 Tasking toolset is fully compatible with RTA-OSEK and does not require modification.

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

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

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

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

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

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

2.2 Assembler

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

Vendor	Tasking
Assembler	TriCore VX-toolset assembler
Version	v2.2r3 patch 1 Build 134.1.2

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

Option	Description
`-C<cpu>`	Selects target CPU
`--silicon-bug=cpu-tc013, cpu-tc048, cpu-tc060, cpu-tc065, cpu-tc068, cpu-tc069, cpu-tc070, cpu-tc071, cpu-tc072, cpu-tc074, cpu-tc081, cpu-tc082, cpu-tc083, cpu-tc094, cpu-tc095, cpu-tc096`	Includes workarounds for Tc17X6 CPU silicon bugs

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

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

Option	Description
`-C<cpu>`	Selects target CPU
`--silicon-bug=cpu-tc013, cpu-tc048, cpu-tc060, cpu-tc065, cpu-tc068, cpu-tc069, cpu-tc070, cpu-tc071, cpu-tc072, cpu-tc074, cpu-tc081, cpu-tc082, cpu-tc083, cpu-tc094, cpu-tc095, cpu-tc096`	Includes workarounds for Tc17X6 CPU silicon bugs

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

Option	Description
`-g`	Enable debugging information

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
.rodata.os*.os_pid	ROM	RTA-OSEK read-only data
.rodata.os*.os_pird	ROM	RTA-OSEK initialization data
.rodata.os*.os_pnird	ROM	RTA-OSEK near initialization data
.text.os*.os_text	RAM	RTA-OSEK library code
.bss.os*.os_pir	RAM	RTA-OSEK initialized data
.data.os*.os_pir	RAM	RTA-OSEK initialized data
.bss.os*.os_pur	RAM	RTA-OSEK uninitialized data
.sbss.os*.os_pnir	RAM	RTA-OSEK A0 uninitialized data
.sdata.os*.os_pnir	RAM	RTA-OSEK A0 initialized data
.sbss.os*.os_cntr	RAM	RTA-OSEK A0 uninitialized data
.sdata.os*.os_cntr	RAM	RTA-OSEK A0 initialized data
.bss.os*.os_trace_ram	RAM	RTA-TRACE data buffer

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 placed in the compiler A0 addressed RAM
*.os_cntr	Must be placed in the compiler A0 addressed RAM

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

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

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. Please refer to the compiler documentation on the use of this section.

2.3.1 Use of SPRAM

The RTA-OSEK Component supports placement of the vector table RTA-OSEK code and RTA-OSEK read only data in SPRAM.

Interrupt vector table placement in RAM

By default the Interrupt vectors are located in ROM. To support RAM location the file `osgen.asm` should be compiled with the command line option `-DOS_VEC_RAM=0`. This define causes the vector table entries to be compiled with the INIT section directive modifier to place it in RAM. Note any Category 1 ISR handlers declared in other C files should include the directive `#pragma section vector_init` to ensure that all interrupt vector entries are placeable in RAM. Please refer to the compiler documentation on the use of this directive.

RTA-OSEK code and constant data placement in RAM

The code and constant data sections of the RTA-OSEK Component are supplied by the run-time libraries and the files `osgen.asm` and `osekdefs.c`. By default these sections are placed in ROM but this can be overridden to place them into RAM.

To place ROM data in the file `osgen.asm` into RAM the command line option for assembling the file should include the command line option `-DOS_DATA_SPRAM`. The inclusion of this define adds the INIT modifier to the section directive so the `os_pid` section will be placed in RAM.

The C code file `osekdefs.c` contains fragments of OS of code and data. To make the code placeable in RAM the command line should include the command `-DOS_CODE_SPRAM`. The presence of this define includes the directive `#pragma section code_init` in the code. To make the ROM data placeable into RAM the command line should include the command `-DOS_DATA_SPRAM` to include the directive `#pragma section const_init` into the code. Please refer to the compiler documentation on the use of these directives.

To support the placement of the RTA-OSEK Component API calls within RAM multiple versions of the libraries are provided. The default versions of the libraries (i.e. all named `rtk_(s/t/e).a` library) configure the code and constant data sections to be placed in ROM. An alternative set of libraries are provided that have been compiled from the same code base but with the `#pragma section code_init` and `#pragma section const_init` directives. These libraries all include `_ram` in the name (i.e. `rtk_s_ram.a` for the standard build library). The `os_text` and sections and `os_pid` sections of these libraries can then be located in RAM.

Placing RTA-OSEK application components in SPRAM

By default the linker will place the initializable code and data section in RAM. Modifications are required to the linker command file to correctly place the sections in SPRAM. To place the vector table in SPRAM the linker command file must include a define of INTTAB with an address in SPRAM (i.e. `#define INTTAB 0xD400B000` for the Tc1796). To place A0 addressed variables in

SPRAM the define A0_START_ADDRESS should be used (i.e. `#define A0_START_ADDRESS 0xD0003700` for the Tc1796). To place the library `osgen.asm` and `osekdefs.c` code and data sections into SPRAM the linker command file should contain a section such as the following:

```
// place the OS in SPRAM
section_layout spe:tc:linear
{
 group os_code (ordered, run_addr=mem:spe:csram)
 {
       select "*.os_text";
       select "*.os_pid";
 }

 group os_data (ordered, run_addr=mem:spe:dsram)
 {
       select "*.os_pnir";
       select "*.os_pur";
       select "*.os_pir";
 }
}
```

2.4 Debugger

ORTI is the OSEK Run-Time Interface that is supported by RTA-OSEK. Support is provided for the debuggers in the following table. Further information about ORTI for RTA-OSEK can be found in the *RTA-OSEK ORTI Guide*.

ORTI Compatible Debuggers
Tasking Crossview Pro 2.2r3 Build 062
Lauterbach Trace32 Build 1194

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model. You can find out more about configuring interrupts for RTA-OSEK in the *RTA-OSEK User Guide*.

3.1.1 Interrupt Levels

In RTA-OSEK interrupts are allocated an Interrupt Priority Level (IPL). This is a processor independent abstraction of the interrupt priorities that are available on the target hardware. You can find out more about IPLs in the *RTA-OSEK User Guide*. The hardware interrupt controller is explained in the *Infineon TriCore User's Manual - System Units*.

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

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. See TriCore Architecture manual for a full explanation.

3.1.3 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Tasking

C compiler can generate appropriate interrupt handling code for a C function decorated with the `__interrupt(<vector>)` `__enable_` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

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

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

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

3.1.6 IO Privilege Mode

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

3.1.7 Interrupt Priorities

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

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

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

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

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

The Category 1 ISRs would be written as

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

and

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

Serialized Category 2 ISRs

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

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

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

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

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

A	2	1	2
Arbitration	Level 2 ordering: B, A.		

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

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

Serialized Category 1 ISRs

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

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

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

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

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

The Category 1 ISRs would be written as

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

and

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

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

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

3.1.8 Default Interrupt

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

3.2 Register Settings

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

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

The RTA-OSEK Component uses the following hardware registers. They 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): 8

Timing

API max usage (bytes): 8

Extended

API max usage (bytes): 16

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

3.3.3 Stack Mode

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

3.3.4 Context Save Areas (CSAs)

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

The worst case CSA usage for Task-to-Task switching in standard build is 6 CSAs. In timing and extended builds the figure is 7 CSAs.

For example, consider an application in standard build that has 10 tasks, the code for each of which requires up to 4 CSAs. The combined CSA requirement for those tasks is at most (4 + 6) * 10 = 100 CSAs. This maximum would occur when all but the highest priority task had been started and then pre-empted by the next, higher priority task and when the highest priority task was currently running.

3.3.5 Call Depth Counter

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

4 Parameters of Implementation

This chapter provides detailed information on the functionality, performance and memory demands of the RTA-OSEK Component.

The RTA-OSEK Component is highly scalable. As a result, different figures will be obtained when your application uses different sets of features. These feature-sets give six classes of RTA-OSEK, depending on whether your application uses events, shared task priorities and/or multiple (queued) task activations. You should identify which class your application belongs to and then use the figures from the appropriate column in the table.

The following hardware was used to take the measurements in this chapter:

Processor	**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	28	28	28	28	28	28
	ROM	162	162	162	162	162	162
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	48	48	48	48	48	48
	ROM	234	234	234	234	234	234
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	66	66	66	66	66	66
	ROM	280	280	280	280	280	280
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

4.2.2 ROM and RAM for OSEK OS Objects

In addition to the base OS overhead, detailed in Section 4.2.1, each OSEK OS object requires ROM and/or RAM. RTA-OSEK provides additional sub-task types for each task type in OSEK (basic and extended), determined by the offline configuration tools. They are as follows:

OSEK Class	Termination	Arithmetic
BCC1	Lightweight	Integer or Floating-Point
BCC1	Heavyweight	Integer or Floating-Point
BCC2	Light or Heavy	Integer or Floating-Point
ECC1	Heavyweight	Integer
ECC1	Heavyweight	Floating-Point
ECC2	Heavyweight	Integer
ECC2	Heavyweight	Floating-Point

The following tables give the ROM and/or RAM requirements (in bytes) for each OS object in the RTA-OSEK Component. (Note that the OSEK COM class was set to CCCA for systems without events, CCCB for systems with events. A default message of size 10 bytes was used for both CCCA and CCCB. The CCCB message size includes queued messages.)

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	36	36	36	36	36	36
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	40	40	40	40	40	40
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	44	52	n/a	44	52
ECC1, Integer task	RAM	n/a	n/a	n/a	28	28	28
	ROM	n/a	n/a	n/a	60	60	60
ECC1, floating-point task	RAM	n/a	n/a	n/a	30	30	30
	ROM	n/a	n/a	n/a	60	60	60
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	30
	ROM	n/a	n/a	n/a	n/a	n/a	68
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	32
	ROM	n/a	n/a	n/a	n/a	n/a	68
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	64	64	64	64	64	64
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	104	104	104	104	104	104
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	34	34	34	34	34	34
Counter	RAM	4	4	4	4	4	4
	ROM	112	112	112	112	112	112
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	52	52	52	52	52	52
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	56	64	n/a	56	64
ECC1, Integer task	RAM	n/a	n/a	n/a	40	40	40
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating-point task	RAM	n/a	n/a	n/a	42	42	42
	ROM	n/a	n/a	n/a	72	72	72
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	42
	ROM	n/a	n/a	n/a	n/a	n/a	80
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	44
	ROM	n/a	n/a	n/a	n/a	n/a	80

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	116	116	116	116	116	116
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	144	144	144	144	144	144
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	34	34	34	34	34	34
Counter	RAM	4	4	4	4	4	4
	ROM	112	112	112	112	112	112
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	16	16	16	16	16	16
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	64	72	n/a	64	72
ECC1, Integer task	RAM	n/a	n/a	n/a	44	44	44
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	46	46	46
	ROM	n/a	n/a	n/a	80	80	80
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	46
	ROM	n/a	n/a	n/a	n/a	n/a	88
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	48
	ROM	n/a	n/a	n/a	n/a	n/a	88
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	128	128	128	128	128	128
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	156	156	156	156	156	156
Resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	38	38	38	38	38	38
Counter	RAM	4	4	4	4	4	4
	ROM	116	116	116	116	116	116
Message	RAM	11	11	11	31	31	31
	ROM	24	24	24	60	60	60
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Arrivalpoint (writable)	RAM	20	20	20	20	20	20
	ROM	20	20	20	20	20	20
Schedule	RAM	20	20	20	20	20	20
	ROM	44	44	44	44	44	44
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

4.2.3 Size of Linkable Modules

The RTA-OSEK Component is demand linked. This means that each API call is placed into a separately linkable module. The following sections list the module sizes (in bytes) for each API call in the 3 RTA-OSEK build types (standard, timing, and extended).

In some cases there are multiple variants of particular API calls. This is because the offline configuration of RTA-OSEK can determine when optimized versions of the API calls can be used. The smallest and fastest call will be selected. In these cases, modules sizes are given for each variant under the particular configuration of the RTA-OSEK Component for which the call is valid.

The call variants are as follows:

Variant	Description
1i	Idle task is only ECC task.
CCCA	OSEK COM class.
CCCB	OSEK COM class.
CLEx	Resource tests in Extended OS Status.
fp	ECC task uses floating-point.
H	Used for heavyweight termination only.

Variant	Description
Hook	Pre- and Post- Task hooks are used.
KL	API is called from OS level.
KL1i	API is called from OS level, idle task is only ECC task.
KL2	Activated taskset has one BCC2 task.
LExt	Used for lightweight termination in Extended Status.
ServiceID	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
Parameters	`ErrorHook` uses `GetServiceID` and `GetServiceParameters`.
NoHook	Pre- and/or Post- Task hooks are not used.
NS	No context switch is possible.
NS1i	No context switch is possible, idle task is only ECC task.
NS2	Activated taskset has one BCC2 task.
NSH	Chain from heavyweight task, not to higher priority.
NSL	Chain from lightweight task, not to higher priority.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.
Task	Resource is used only by tasks.

Standard

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	104	164	198	110	168	232
	NS		88	148	182	94	154	216
	KL	2	46	106	140	52	112	174

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	14	14	14	14	14	14
ChainTask	SWL	1, 8	106	166	204	112	172	236
	SWH	1, 9	132	194	230	138	200	260
	NSL	8	106	166	204	112	172	236
	NSH	9	120	182	218	126	188	248
Schedule			82	82	110	82	82	110
GetTaskID			30	30	30	30	30	30
GetTaskState			68	68	68	80	80	80
EnableAllInterrupts			30	30	30	30	30	30
DisableAllInterrupts			30	30	30	30	30	30
ResumeAllInterrupts			46	46	46	46	46	46
SuspendAllInterrupts			46	46	46	46	46	46
ResumeOSInterrupts			46	46	46	46	46	46
SuspendOSInterrupts			50	50	50	50	50	50
GetResource	Task	7	22	22	28	22	22	28
	Combined	6	64	64	64	64	64	64
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	64	64	64	64	64	64
	Combined	6	158	158	158	158	158	158
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	112	112	188
	NS		n/a	n/a	n/a	94	94	170
	NS1i	10	n/a	n/a	n/a	54	n/a	n/a
	KL	2	n/a	n/a	n/a	62	62	140
	KL1i	2, 10	n/a	n/a	n/a	24	n/a	n/a
ClearEvent			n/a	n/a	n/a	60	60	60
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	220	220	434
	fp	11	n/a	n/a	n/a	250	250	504
	1i	10	n/a	n/a	n/a	18	n/a	n/a
GetAlarmBase			52	52	52	52	52	52
GetAlarm			80	80	80	80	80	80
SetRelAlarm			110	110	110	110	110	110
SetAbsAlarm			126	126	126	126	126	126
CancelAlarm			72	72	72	72	72	72
InitCounter			54	54	54	54	54	54

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			68	68	68	68	68	68
osek_tick_alarm	<default>		60	60	60	60	60	60
	KL	2	32	32	32	32	32	32
osek_incr_counter			92	92	92	92	92	92
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			244	244	244	244	244	244
ShutdownOS	NoHook	12	36	36	36	36	36	36
	Hook	13	50	50	50	50	50	50
InitCOM			6	6	6	6	6	6
CloseCOM			6	6	6	6	6	6
StartCOM			28	28	28	28	28	28
StopCOM			18	18	18	18	18	18
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	54	54	54	154	154	154
	CCCB	15	154	154	154	154	154	154
GetMessageResource			34	34	34	34	34	34
ReleaseMessageResource			34	34	34	34	34	34
GetMessageStatus			42	42	42	42	42	42
SendMessage	SW CCCA	1, 14	78	78	78	202	202	202
	SW CCCB	1, 15	188	188	188	202	202	202
	NS CCCA	14	78	78	78	202	202	202
	NS CCCB	15	188	188	188	202	202	202
	KL CCCA	2, 14	48	48	48	172	172	172
	KL CCCB	2, 15	158	158	158	172	172	172
main_dispatch	NoHook	12	122	122	160	122	122	160
	Hook	13	158	158	196	158	158	196
sub_dispatch	B1LF	19	20	20	20	20	20	20
	B1HI	20	90	90	90	90	90	90
	B1HF	21	96	96	96	96	96	96
	B2LI	22	n/a	78	102	n/a	78	102
	B2LF	23	n/a	86	110	n/a	86	110
	B2HI	24	n/a	170	240	n/a	170	240
	B2HF	25	n/a	178	248	n/a	178	248
	E1HI	26	n/a	n/a	n/a	382	382	452
	E1HF	27	n/a	n/a	n/a	390	390	458
	E2HI	28	n/a	n/a	n/a	n/a	n/a	452

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	E2HF	29	n/a	n/a	n/a	n/a	n/a	458
ErrorHook support		16	38	38	38	38	38	38
	ServiceID	17	46	46	46	46	46	46
	Parameters	18	66	66	66	66	66	66
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	106	222	252	128	256	328
	NS		90	206	236	112	240	314
	KL	2	46	164	194	68	198	272
ChainTaskset	SWL	1, 8	124	248	282	130	274	348
	SWH	1, 9	150	282	314	156	306	376
	NSL	8	124	248	282	130	274	348
	NSH	9	138	270	302	144	294	364
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			42	42	42	42	42	42
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			52	52	52	52	52	52
TestSubTaskset			44	44	44	44	44	44
TestEquivalentTaskset			42	42	42	42	42	42
TickSchedule	SW	1	158	120	120	120	120	120
	NS		142	96	96	96	96	96
	KL	2	110	64	64	64	64	64
AdvanceSchedule	SW	1	150	106	106	106	106	106
	NS		130	90	90	90	90	90
	KL	2	98	58	58	58	58	58
StartSchedule			68	68	68	68	68	68
StopSchedule			54	54	54	54	54	54
GetScheduleStatus			82	82	82	82	82	82
GetScheduleValue			66	66	66	66	66	66
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

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestArrivalpointWritable			36	36	36	36	36	36
GetExecutionTime			6	6	6	6	6	6
GetLargestExecutionTime			8	8	8	8	8	8
ResetLargestExecutionTime			6	6	6	6	6	6
GetStackOffset			18	18	18	18	18	18
Floating point support			40	40	40	40	40	40
Interrupt support			56	56	56	56	56	56
Context save			52	52	52	52	52	52
Context restore			164	164	164	164	164	164

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	104	164	198	110	168	232
	NS		88	148	182	94	154	216
	KL	2	46	106	140	52	112	174
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	14	14	14	14	14	14
ChainTask	SWL	1, 8	106	166	204	112	172	236
	SWH	1, 9	132	194	230	138	200	260
	NSL	8	106	166	204	112	172	236
	NSH	9	120	182	218	126	188	248
Schedule			104	104	132	104	104	132
GetTaskID			30	30	30	30	30	30
GetTaskState			68	68	68	80	80	80
EnableAllInterrupts			30	30	30	30	30	30
DisableAllInterrupts			30	30	30	30	30	30
ResumeAllInterrupts			46	46	46	46	46	46
SuspendAllInterrupts			46	46	46	46	46	46
ResumeOSInterrupts			46	46	46	46	46	46
SuspendOSInterrupts			50	50	50	50	50	50
GetResource	Task	7	22	22	28	22	22	28

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Combined	6	64	64	64	64	64	64
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	88	88	88	88	88	88
	Combined	6	204	204	204	204	204	204
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	112	112	188
	NS		n/a	n/a	n/a	94	94	170
	NS1i	10	n/a	n/a	n/a	54	n/a	n/a
	KL	2	n/a	n/a	n/a	62	62	140
	KL1i	2, 10	n/a	n/a	n/a	24	n/a	n/a
ClearEvent			n/a	n/a	n/a	60	60	60
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	320	320	536
	fp	11	n/a	n/a	n/a	342	342	586
	1i	10	n/a	n/a	n/a	108	n/a	n/a
GetAlarmBase			52	52	52	52	52	52
GetAlarm			80	80	80	80	80	80
SetRelAlarm			110	110	110	110	110	110
SetAbsAlarm			126	126	126	126	126	126
CancelAlarm			72	72	72	72	72	72
InitCounter			54	54	54	54	54	54
GetCounterValue			68	68	68	68	68	68
osek_tick_alarm	<default>		60	60	60	60	60	60
	KL	2	32	32	32	32	32	32
osek_incr_counter			92	92	92	92	92	92
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			314	314	314	314	314	314
ShutdownOS	NoHook	12	36	36	36	36	36	36
	Hook	13	50	50	50	50	50	50
InitCOM			6	6	6	6	6	6
CloseCOM			6	6	6	6	6	6
StartCOM			28	28	28	28	28	28
StopCOM			18	18	18	18	18	18
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	54	54	54	154	154	154
	CCCB	15	154	154	154	154	154	154

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageResource			34	34	34	34	34	34
ReleaseMessageResource			34	34	34	34	34	34
GetMessageStatus			42	42	42	42	42	42
SendMessage	SW CCCA	1, 14	78	78	78	202	202	202
	SW CCCB	1, 15	188	188	188	202	202	202
	NS CCCA	14	78	78	78	202	202	202
	NS CCCB	15	188	188	188	202	202	202
	KL CCCA	2, 14	48	48	48	172	172	172
	KL CCCB	2, 15	158	158	158	172	172	172
main_dispatch	NoHook	12	188	188	228	188	188	228
	Hook	13	230	230	270	230	230	270
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	90	90	90	90	90	90
	B1HF	21	100	100	100	100	100	100
	B2LI	22	n/a	52	86	n/a	52	86
	B2LF	23	n/a	60	94	n/a	60	94
	B2HI	24	n/a	132	220	n/a	132	220
	B2HF	25	n/a	140	228	n/a	140	228
	E1HI	26	n/a	n/a	n/a	398	398	466
	E1HF	27	n/a	n/a	n/a	406	406	472
	E2HI	28	n/a	n/a	n/a	n/a	n/a	466
	E2HF	29	n/a	n/a	n/a	n/a	n/a	472
ErrorHook support		16	38	38	38	38	38	38
	ServiceID	17	46	46	46	46	46	46
	Parameters	18	66	66	66	66	66	66
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	74	74	74	74	74	74
ActivateTaskset	SW	1	106	222	252	128	256	328
	NS		90	206	236	112	240	314
	KL	2	46	164	194	68	198	272
ChainTaskset	SWL	1, 8	124	248	282	130	274	348
	SWH	1, 9	150	282	314	156	306	376
	NSL	8	124	248	282	130	274	348
	NSH	9	138	270	302	144	294	364
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			42	42	42	42	42	42

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			52	52	52	52	52	52
TestSubTaskset			44	44	44	44	44	44
TestEquivalentTaskset			42	42	42	42	42	42
TickSchedule	SW	1	158	120	120	120	120	120
	NS		142	96	96	96	96	96
	KL	2	110	64	64	64	64	64
AdvanceSchedule	SW	1	150	106	106	106	106	106
	NS		130	90	90	90	90	90
	KL	2	98	58	58	58	58	58
StartSchedule			68	68	68	68	68	68
StopSchedule			54	54	54	54	54	54
GetScheduleStatus			82	82	82	82	82	82
GetScheduleValue			66	66	66	66	66	66
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			36	36	36	36	36	36
GetExecutionTime			106	106	106	106	106	106
GetLargestExecutionTime			14	14	14	14	14	14
ResetLargestExecutionTime			12	12	12	12	12	12
GetStackOffset			18	18	18	18	18	18
Floating point support			40	40	40	40	40	40
Interrupt support			222	222	222	222	222	222
Context save			52	52	52	52	52	52
Context restore			164	164	164	164	164	164

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	214	276	314	228	282	344
	NS		250	312	350	264	318	380
	KL	2	130	192	230	144	198	260
TerminateTask	LExt	3	94	94	94	94	94	94
	H	5	118	118	118	118	118	118
ChainTask	SWL	1, 8	246	312	350	262	318	382
	SWH	1, 9	278	348	384	294	354	412
	NSL	8	284	350	388	300	356	420
	NSH	9	310	380	416	326	386	444
Schedule			222	222	250	222	222	250
GetTaskID			46	46	46	46	46	46
GetTaskState			208	208	208	212	212	212
EnableAllInterrupts			42	42	42	42	42	42
DisableAllInterrupts			42	42	42	42	42	42
ResumeAllInterrupts			86	86	86	86	86	86
SuspendAllInterrupts			58	58	58	58	58	58
ResumeOSInterrupts			86	86	86	86	86	86
SuspendOSInterrupts			66	66	66	66	66	66
GetResource	Task	7	324	324	302	324	324	302
	Combined	6	306	306	306	306	306	306
	CLEx	3	282	282	282	282	282	282
ReleaseResource	Task	7	306	306	306	306	306	306
	Combined	6	408	408	408	408	408	408
	CLEx	3	268	268	268	268	268	268
SetEvent	SW	1	n/a	n/a	n/a	270	270	346
	NS		n/a	n/a	n/a	304	304	380
	NS1i	10	n/a	n/a	n/a	204	n/a	n/a
	KL	2	n/a	n/a	n/a	186	186	264
	KL1i	2, 10	n/a	n/a	n/a	136	n/a	n/a
ClearEvent			n/a	n/a	n/a	126	126	126
GetEvent			n/a	n/a	n/a	138	138	138
WaitEvent	<default>		n/a	n/a	n/a	422	422	628
	fp	11	n/a	n/a	n/a	450	450	680

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	216	n/a	n/a
GetAlarmBase			170	170	170	170	170	170
GetAlarm			154	154	154	154	154	154
SetRelAlarm			208	208	208	208	208	208
SetAbsAlarm			228	228	228	228	228	228
CancelAlarm			144	144	144	144	144	144
InitCounter			214	214	214	214	214	214
GetCounterValue			188	188	188	188	188	188
osek_tick_alarm	<default>		100	100	100	100	100	100
	KL	2	32	32	32	32	32	32
osek_incr_counter			92	92	92	92	92	92
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			330	330	330	330	330	330
ShutdownOS	NoHook	12	46	46	46	46	46	46
	Hook	13	60	60	60	60	60	60
InitCOM			6	6	6	6	6	6
CloseCOM			6	6	6	6	6	6
StartCOM			46	46	46	46	46	46
StopCOM			46	46	46	46	46	46
ReadFlag			32	32	32	32	32	32
ResetFlag			36	36	36	36	36	36
ReceiveMessage	CCCA	14	150	150	150	250	250	250
	CCCB	15	250	250	250	250	250	250
GetMessageResource			76	76	76	76	76	76
ReleaseMessageResource			76	76	76	76	76	76
GetMessageStatus			82	82	82	82	82	82
SendMessage	SW CCCA	1, 14	188	188	188	306	306	306
	SW CCCB	1, 15	292	292	292	306	306	306
	NS CCCA	14	188	188	188	306	306	306
	NS CCCB	15	292	292	292	306	306	306
	KL CCCA	2, 14	122	122	122	240	240	240
	KL CCCB	2, 15	226	226	226	240	240	240
main_dispatch	NoHook	12	188	188	228	188	188	228
	Hook	13	230	230	270	230	230	270
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	90	90	90	90	90	90
	B1HF	21	100	100	100	100	100	100

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	B2LI	22	n/a	50	84	n/a	50	84
	B2LF	23	n/a	58	92	n/a	58	92
	B2HI	24	n/a	130	216	n/a	130	216
	B2HF	25	n/a	138	224	n/a	138	224
	E1HI	26	n/a	n/a	n/a	398	398	466
	E1HF	27	n/a	n/a	n/a	406	406	472
	E2HI	28	n/a	n/a	n/a	n/a	n/a	466
	E2HF	29	n/a	n/a	n/a	n/a	n/a	472
ErrorHook support		16	110	110	110	110	110	110
	ServiceID	17	118	118	118	118	118	118
	Parameters	18	138	138	138	138	138	138
validity_checks		3	22	22	22	22	22	22
Timing_dispatch		4	80	80	80	80	80	80
Timing_termination		4	74	74	74	74	74	74
ActivateTaskset	SW	1	296	380	424	346	418	468
	NS		338	410	454	380	448	502
	KL	2	214	300	344	262	336	396
ChainTaskset	SWL	1, 8	348	432	470	386	456	512
	SWH	1, 9	386	494	526	424	518	566
	NSL	8	406	480	518	432	502	564
	NSH	9	432	528	560	458	550	606
GetTasksetRef			108	108	108	108	108	108
MergeTaskset			294	294	294	294	294	294
AssignTaskset			204	204	204	204	204	204
RemoveTaskset			304	304	304	304	304	304
TestSubTaskset			300	300	300	300	300	300
TestEquivalentTaskset			298	298	298	298	298	298
TickSchedule	SW	1	310	238	238	238	238	238
	NS		342	298	298	298	298	298
	KL	2	236	170	170	170	170	170
AdvanceSchedule	SW	1	312	246	246	246	246	246
	NS		344	296	296	296	296	296
	KL	2	244	176	176	176	176	176
StartSchedule			224	224	224	224	224	224
StopSchedule			186	186	186	186	186	186
GetScheduleStatus			216	216	216	216	216	216
GetScheduleValue			176	176	176	176	176	176

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleNext			90	90	90	90	90	90
SetScheduleNext			178	178	178	178	178	178
GetArrivalpointDelay			128	128	128	128	128	128
SetArrivalpointDelay			144	144	144	144	144	144
GetArrivalpointTasksetRef			126	126	126	126	126	126
GetArrivalpointNext			128	128	128	128	128	128
SetArrivalpointNext			216	216	216	216	216	216
TestArrivalpointWritable			144	144	144	144	144	144
GetExecutionTime			156	156	156	156	156	156
GetLargestExecutionTime			94	94	94	94	94	94
ResetLargestExecutionTime			88	88	88	88	88	88
GetStackOffset			18	18	18	18	18	18
Floating point support			40	40	40	40	40	40
Interrupt support			222	222	222	222	222	222
Context save			52	52	52	52	52	52
Context restore			164	164	164	164	164	164

Notes

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

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

4.2.4 Reserved Hardware Resources

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

4.3 Performance

4.3.1 Execution Times for RTA-OSEK API Calls

The following tables give the execution time (in CPU cycles) for each API call. (Note that: (1) the OSEK COM class was set to CCCA for systems without events and to CCCB for systems with events; (2) `ShutdownOS()` enters an infinite loop; the execution time for `ShutdownOS()` reported below is the time up to the point at which `ShutdownOS()` calls `ShutdownHook()`).

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	50	70	83	51	63	85
	NS	41	61	75	43	55	78
	KL	33	53	66	34	48	68
TerminateTask	LExt	0	0	0	0	0	0
	H	152	154	153	152	152	154
ChainTask	SWL	165	185	218	191	201	246
	SWH	257	277	310	282	296	338
	NSL	165	186	218	191	201	247
	NSH	254	277	308	280	292	337
Schedule	SW	47	45	54	44	44	54
GetTaskID		32	32	32	50	50	50
GetTaskState		43	43	43	65	66	65
EnableAllInterrupts		33	33	33	33	33	33
DisableAllInterrupts		28	28	28	28	28	28
ResumeAllInterrupts		39	39	39	39	39	39
SuspendAllInterrupts		30	30	30	30	30	30
ResumeOSInterrupts		39	39	39	39	39	39
SuspendOSInterrupts		30	30	30	30	30	30
GetResource	Task	33	33	33	33	33	33
	Combined	38	38	38	38	38	38
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	39	39	40	39	38	38
	Combined	52	52	52	52	52	52
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	49	49	51
	NS	n/a	n/a	n/a	44	48	47
	KL	n/a	n/a	n/a	37	37	37
ClearEvent		n/a	n/a	n/a	34	35	34
GetEvent		n/a	n/a	n/a	25	25	25
WaitEvent	<default>	n/a	n/a	n/a	314	313	337
	fp	n/a	n/a	n/a	319	319	341
GetAlarmBase		38	39	39	37	37	37
GetAlarm		44	44	44	44	44	44

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		49	49	49	50	50	49
SetAbsAlarm		48	48	48	48	48	49
CancelAlarm		38	37	37	38	37	37
InitCounter		36	36	36	37	37	37
GetCounterValue		42	42	42	42	42	42
osek_tick_alarm	<default>	42	44	44	43	42	43
	KL	35	35	35	34	34	34
osek_incr_counter		16	16	16	16	16	16
GetActiveApplicationMode		14	14	14	14	14	14
StartOS		1574	1574	1574	1572	1574	1573
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	23	23	23	23	23	23
InitCOM		18	18	18	18	18	18
CloseCOM		18	18	18	19	23	19
StartCOM		37	37	37	103	103	103
StopCOM		22	22	22	21	21	21
ReadFlag		n/a	n/a	n/a	19	19	19
ResetFlag		n/a	n/a	n/a	17	17	17
ReceiveMessage		31	31	31	117	118	117
GetMessageResource		n/a	n/a	n/a	53	53	52
ReleaseMessageResource		n/a	n/a	n/a	60	59	59
GetMessageStatus		n/a	n/a	n/a	35	35	35
SendMessage	SW	85	104	117	169	192	202
	NS	77	98	110	159	171	194
	KL	56	76	89	140	152	175
ActivateTaskset	SW	48	470	451	53	436	519
	NS	40	460	473	46	396	483
	KL	30	449	462	35	387	501
	SW2	48	470	451	53	436	519
	NS2	40	460	473	46	396	482
	KL2	30	449	462	35	387	501
ChainTaskset	SWL	174	656	750	199	673	681
	SWH	262	745	778	287	640	804
	NSL	174	656	750	199	674	682
	NSH	259	744	776	284	636	802
GetTasksetRef		43	43	43	44	44	45

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		53	53	56	54	54	54
AssignTaskset		40	40	40	40	40	40
RemoveTaskset		52	52	52	52	52	52
TestSubTaskset		53	53	53	53	53	53
TestEquivalentTaskset		50	50	50	50	50	50
TickSchedule	SW	71	494	508	80	447	560
	NS	62	484	497	70	437	551
	KL	53	475	488	61	428	541
	SW2	71	494	508	80	433	547
	NS2	62	485	497	70	422	537
	KL2	53	475	488	61	413	528
AdvanceSchedule	SW	68	493	504	77	444	557
	NS	59	481	494	68	434	547
	KL	52	474	487	60	427	540
	SW2	68	492	504	77	429	545
	NS2	59	481	494	68	419	534
	KL2	52	474	487	60	412	527
StartSchedule		64	65	64	64	64	64
StopSchedule		59	59	59	59	59	59
GetScheduleStatus		65	65	65	65	65	65
GetScheduleValue		60	61	61	60	61	60
GetScheduleNext		44	44	44	44	44	44
SetScheduleNext		41	41	41	41	41	41
GetArrivalpointDelay		42	42	42	42	42	42
SetArrivalpointDelay		39	39	39	39	39	39
GetArrivalpointTasksetRef		40	40	40	40	40	40
GetArrivalpointNext		40	40	40	40	40	41
SetArrivalpointNext		39	39	39	39	39	39
TestArrivalpointWritable		50	50	50	50	50	50
GetExecutionTime		19	19	19	19	19	19
GetLargestExecutionTime		24	24	24	24	24	24
ResetLargestExecutionTime		18	18	18	18	18	18
GetStackOffset		25	25	25	25	25	25

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	50	70	83	51	63	85
	NS	42	61	74	44	55	77
	KL	33	53	66	34	46	69
TerminateTask	LExt	0	0	0	0	0	0
	H	274	273	273	273	275	273
ChainTask	SWL	269	290	328	297	310	356
	SWH	371	392	428	399	413	459
	NSL	269	290	328	297	310	356
	NSH	369	389	426	397	410	456
Schedule	SW	44	45	54	44	45	54
GetTaskID		32	32	36	50	50	50
GetTaskState		43	43	43	65	65	66
EnableAllInterrupts		33	33	33	33	33	33
DisableAllInterrupts		28	28	28	28	28	28
ResumeAllInterrupts		39	39	39	39	39	39
SuspendAllInterrupts		30	32	32	30	30	30
ResumeOSInterrupts		39	39	39	39	39	40
SuspendOSInterrupts		30	32	32	30	30	30
GetResource	Task	33	33	33	33	33	33
	Combined	38	38	38	38	38	38
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	37	37	36	35	35	35
	Combined	51	52	52	51	53	51
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	49	49	52
	NS	n/a	n/a	n/a	45	44	44
	KL	n/a	n/a	n/a	38	37	37
ClearEvent		n/a	n/a	n/a	34	34	34
GetEvent		n/a	n/a	n/a	25	25	25
WaitEvent	<default>	n/a	n/a	n/a	421	422	443
	fp	n/a	n/a	n/a	429	429	447
GetAlarmBase		41	39	38	37	37	37
GetAlarm		45	44	44	44	45	45

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		49	49	48	49	50	49
SetAbsAlarm		48	48	48	48	48	48
CancelAlarm		37	37	37	37	38	38
InitCounter		36	36	36	37	37	37
GetCounterValue		42	42	42	42	42	42
osek_tick_alarm	<default>	42	43	43	44	43	43
	KL	35	35	35	34	34	34
osek_incr_counter		16	16	16	16	16	16
GetActiveApplicationMode		14	14	14	14	14	14
StartOS		4185	4185	4161	4185	4161	4160
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	23	23	23	23	23	23
InitCOM		18	18	18	18	18	18
CloseCOM		18	18	18	19	19	19
StartCOM		37	37	37	103	103	103
StopCOM		23	22	23	21	21	21
ReadFlag		n/a	n/a	n/a	19	19	19
ResetFlag		n/a	n/a	n/a	17	17	17
ReceiveMessage		31	31	31	116	116	117
GetMessageResource		n/a	n/a	n/a	52	52	52
ReleaseMessageResource		n/a	n/a	n/a	58	57	58
GetMessageStatus		n/a	n/a	n/a	35	35	35
SendMessage	SW	84	104	117	169	180	202
	NS	78	97	110	160	172	193
	KL	56	76	89	139	151	176
ActivateTaskset	SW	48	468	451	54	403	487
	NS	40	460	443	46	429	511
	KL	30	449	463	36	386	499
	SW2	48	468	451	54	403	487
	NS2	40	460	442	46	429	511
	KL2	30	449	464	36	386	499
ChainTaskset	SWL	278	761	798	305	751	764
	SWH	376	865	896	405	758	924
	NSL	278	761	798	305	814	795
	NSH	374	889	894	400	818	893
GetTasksetRef		43	43	43	44	44	44

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		54	53	53	54	54	54
AssignTaskset		40	40	40	40	40	40
RemoveTaskset		52	52	52	51	52	52
TestSubTaskset		53	53	53	53	53	53
TestEquivalentTaskset		50	50	50	50	51	50
TickSchedule	SW	71	494	507	81	444	558
	NS	63	484	498	71	435	549
	KL	53	475	489	62	427	539
	SW2	71	494	507	81	429	545
	NS2	63	484	498	71	421	536
	KL2	53	475	488	62	411	526
AdvanceSchedule	SW	68	490	504	78	442	555
	NS	59	483	495	68	433	545
	KL	51	474	489	62	425	538
	SW2	68	493	504	78	427	542
	NS2	59	482	495	68	418	533
	KL2	51	475	489	62	410	525
StartSchedule		64	64	64	64	64	64
StopSchedule		59	59	59	59	59	59
GetScheduleStatus		65	65	65	66	65	65
GetScheduleValue		60	60	60	60	61	61
GetScheduleNext		44	44	44	44	44	44
SetScheduleNext		41	41	41	41	41	41
GetArrivalpointDelay		42	42	42	42	42	42
SetArrivalpointDelay		39	39	39	39	39	39
GetArrivalpointTasksetRef		40	40	40	40	40	40
GetArrivalpointNext		40	40	40	40	40	40
SetArrivalpointNext		39	39	39	39	39	39
TestArrivalpointWritable		50	50	50	50	50	50
GetExecutionTime		61	61	61	61	62	61
GetLargestExecutionTime		26	26	26	26	26	26
ResetLargestExecutionTime		24	24	24	24	24	24
GetStackOffset		22	22	22	22	22	22

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	157	178	193	162	174	192
	NS	177	197	210	181	191	210
	KL	134	155	169	138	149	167
TerminateTask	LExt	229	230	228	228	229	228
	H	323	323	322	323	323	322
ChainTask	SWL	416	439	475	446	457	504
	SWH	517	539	575	546	561	603
	NSL	440	462	497	470	480	527
	NSH	538	562	594	567	579	623
Schedule	SW	75	75	84	75	74	83
GetTaskID		34	34	34	53	53	53
GetTaskState		164	164	162	184	181	183
EnableAllInterrupts		35	36	36	35	36	35
DisableAllInterrupts		32	31	32	31	32	32
ResumeAllInterrupts		47	47	47	47	48	47
SuspendAllInterrupts		32	32	32	32	33	32
ResumeOSInterrupts		47	47	47	47	48	47
SuspendOSInterrupts		32	32	32	32	33	32
GetResource	Task	242	242	145	266	266	169
	Combined	131	131	132	156	156	156
	CLEx	185	183	183	209	209	207
ReleaseResource	Task	136	135	136	161	161	160
	Combined	138	138	140	163	163	163
	CLEx	153	153	152	176	176	175
SetEvent	SW	n/a	n/a	n/a	170	172	172
	NS	n/a	n/a	n/a	176	177	177
	KL	n/a	n/a	n/a	151	150	150
ClearEvent		n/a	n/a	n/a	47	47	48
GetEvent		n/a	n/a	n/a	138	137	137
WaitEvent	<default>	n/a	n/a	n/a	508	509	531
	fp	n/a	n/a	n/a	514	516	535
GetAlarmBase		120	123	122	121	122	123
GetAlarm		123	124	123	124	123	123

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		134	135	134	133	133	133
SetAbsAlarm		133	132	132	133	132	132
CancelAlarm		115	115	115	115	115	115
InitCounter		184	184	184	182	182	182
GetCounterValue		118	118	119	116	116	116
osek_tick_alarm	<default>	57	57	57	57	57	57
	KL	33	33	33	33	33	33
osek_incr_counter		15	15	15	15	15	15
GetActiveApplicationMode		14	14	14	14	14	14
StartOS		4332	4335	4332	4332	4333	4333
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	24	24	24	24	24	24
InitCOM		18	18	18	18	18	18
CloseCOM		18	18	18	19	19	19
StartCOM		44	44	44	110	110	112
StopCOM		30	30	30	29	29	29
ReadFlag		n/a	n/a	n/a	32	32	32
ResetFlag		n/a	n/a	n/a	30	30	30
ReceiveMessage		92	96	93	178	177	178
GetMessageResource		n/a	n/a	n/a	235	235	236
ReleaseMessageResource		n/a	n/a	n/a	222	223	221
GetMessageStatus		n/a	n/a	n/a	83	83	83
SendMessage	SW	255	276	291	340	353	370
	NS	277	296	309	360	370	388
	KL	214	235	249	299	309	327
ActivateTaskset	SW	416	501	545	364	501	525
	NS	396	483	587	376	484	565
	KL	326	445	522	335	445	532
	SW2	416	501	544	365	501	525
	NS2	396	483	587	376	484	565
	KL2	326	445	522	335	445	531
ChainTaskset	SWL	653	734	831	647	754	862
	SWH	722	866	931	747	856	961
	NSL	662	752	849	663	805	882
	NSH	728	884	976	797	871	982
GetTasksetRef		148	147	147	146	146	146

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		118	116	117	117	116	117
AssignTaskset		90	89	90	89	87	89
RemoveTaskset		113	115	113	114	115	113
TestSubTaskset		116	116	117	118	117	118
TestEquivalentTaskset		115	115	115	114	113	115
TickSchedule	SW	133	521	596	409	532	614
	NS	149	557	634	446	568	654
	KL	85	494	570	384	504	588
	SW2	133	521	596	409	521	605
	NS2	149	557	634	446	557	644
	KL2	85	494	570	384	493	579
AdvanceSchedule	SW	135	541	619	432	553	634
	NS	152	560	634	446	567	651
	KL	106	515	591	404	525	609
	SW2	135	541	618	433	542	625
	NS2	152	560	634	446	556	642
	KL2	106	515	591	404	514	600
StartSchedule		107	106	106	107	106	107
StopSchedule		93	94	94	94	94	94
GetScheduleStatus		99	99	99	99	99	99
GetScheduleValue		91	91	91	91	92	92
GetScheduleNext		62	62	63	62	62	61
SetScheduleNext		90	90	89	90	90	89
GetArrivalpointDelay		71	72	71	71	71	72
SetArrivalpointDelay		74	74	73	74	74	74
GetArrivalpointTasksetRef		61	62	61	61	61	61
GetArrivalpointNext		61	60	61	60	61	60
SetArrivalpointNext		96	95	96	94	96	96
TestArrivalpointWritable		69	69	69	69	69	70
GetExecutionTime		79	80	79	79	80	79
GetLargestExecutionTime		122	122	121	122	121	122
ResetLargestExecutionTime		116	116	116	115	116	115
GetStackOffset		22	22	22	22	22	22

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	31	31	31	31	31	31
	Cat 2	30	30	30	30	30	30

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	31	31	31	31	31	31
	Cat 2	153	154	153	153	153	153

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	31	31	31	31	31	31
	Cat 2	153	153	153	153	153	153

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.

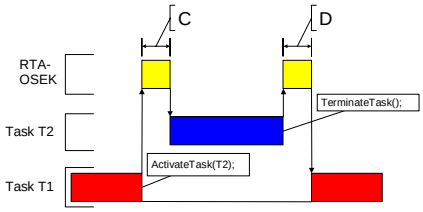

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

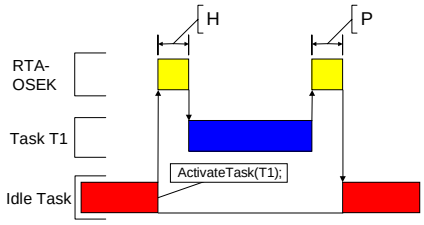

Figure 3: Task Activation from Idle Task

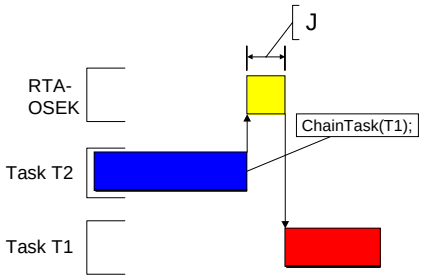

Figure 2: Task Chaining

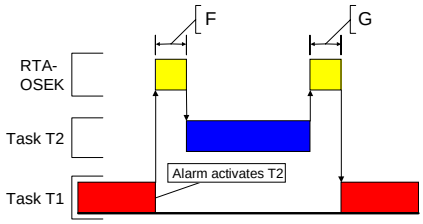

Figure 4: Task Activation from an Alarm

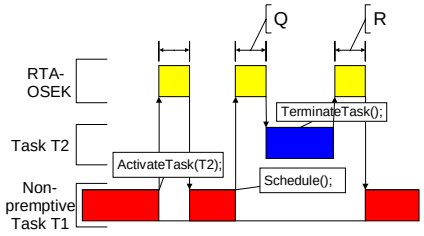

Figure 5: Non-Premptive Task Calls Schedule()

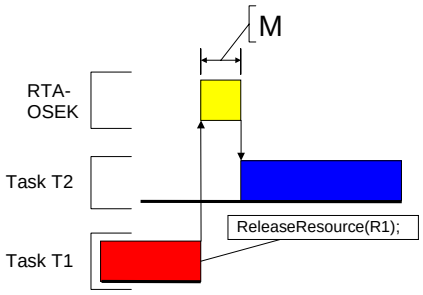

Figure 6: Blocked Task Activated by ReleaseResource()

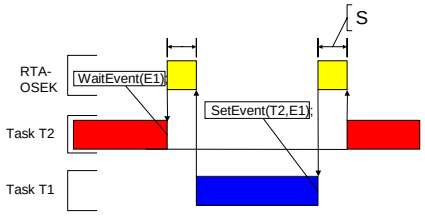

Figure 7: Waiting Task Activated by SetEvent()

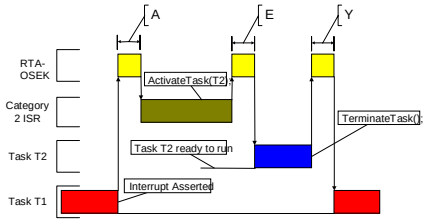

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	64	84	94	64	83	95
Figure 1: D	Heavy, Basic/Extended	153	171	180	176	176	189
ChainTask	Light, Basic	83	113	148	86	114	157
Figure 2: J	Heavy, Basic/Extended	340	389	432	365	394	451
Pre-emption	Light, Basic	80	109	145	82	109	155
Figure 1: C	Heavy, Basic/Extended	165	185	220	191	203	248
From idle task	Light, Basic	129	158	194	131	158	203
Figure 3: H	Heavy, Basic/Extended	214	234	269	240	253	298
Triggered by alarm	Light, Basic	119	151	188	122	148	195
Figure 4: F	Heavy, Basic/Extended	207	230	265	235	246	292
Schedule	Light, Basic	71	79	111	70	79	114
Figure 5: Q	Heavy, Basic/Extended	156	155	186	179	179	211
Release resource	Light, Basic	77	85	108	76	85	108
Figure 6: M	Heavy, Basic/Extended	162	161	183	185	185	208
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	256	256	309
From category 2 ISR	Light, Basic	71	80	103	71	81	103
Figure 8: E	Heavy, Basic/Extended	157	157	179	181	182	204

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	175	189	201	173	189	200
Figure 1: D	Heavy, Basic/Extended	275	288	300	294	296	306
ChainTask	Light, Basic	192	218	255	193	218	263
Figure 2: J	Heavy, Basic/Extended	578	620	666	600	627	681
Pre-emption	Light, Basic	174	201	239	177	200	249
Figure 1: C	Heavy, Basic/Extended	257	278	316	285	297	351
From idle task	Light, Basic	223	253	288	225	249	298
Figure 3: H	Heavy, Basic/Extended	306	327	365	334	349	396
Triggered by alarm	Light, Basic	214	241	279	219	240	290
Figure 4: F	Heavy, Basic/Extended	298	320	358	329	339	390
Schedule	Light, Basic	164	171	205	164	170	207
Figure 5: Q	Heavy, Basic/Extended	248	249	282	273	273	310
Release resource	Light, Basic	168	178	201	168	174	201
Figure 6: M	Heavy, Basic/Extended	251	252	277	277	277	305
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	334	334	392
From category 2 ISR	Light, Basic	271	277	305	269	278	306
Figure 8: E	Heavy, Basic/Extended	352	353	380	377	379	408

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	228	244	254	227	242	255
Figure 1: D	Heavy, Basic/Extended	325	337	347	344	344	356
ChainTask	Light, Basic	339	368	406	343	368	413
Figure 2: J	Heavy, Basic/Extended	773	814	863	797	821	876
Pre-emption	Light, Basic	277	303	345	281	305	349
Figure 1: C	Heavy, Basic/Extended	360	380	424	390	402	447
From idle task	Light, Basic	326	352	394	332	354	398
Figure 3: H	Heavy, Basic/Extended	410	433	469	439	451	496
Triggered by alarm	Light, Basic	334	361	402	339	362	406
Figure 4: F	Heavy, Basic/Extended	419	440	479	449	461	506
Schedule	Light, Basic	191	196	232	190	195	231
Figure 5: Q	Heavy, Basic/Extended	274	273	307	299	298	334
Release resource	Light, Basic	254	259	288	279	285	312
Figure 6: M	Heavy, Basic/Extended	337	340	363	388	388	414
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	487	491	545
From category 2 ISR	Light, Basic	291	296	323	289	296	323
Figure 8: E	Heavy, Basic/Extended	372	371	396	396	397	424

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	28	29	n/a	28	29
BCC2 heavyweight, floating-point	n/a	28	29	n/a	28	29
ECC1 heavyweight, integer	n/a	n/a	n/a	30	30	31
ECC1 heavyweight, floating-point	n/a	n/a	n/a	30	30	31
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	31
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	31
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	13	13	13	13	13	13
BCC1 lightweight, floating-point	14	14	14	14	14	14
BCC1 heavyweight, integer	29	29	29	29	29	29
BCC1 heavyweight, floating-point	29	29	29	29	29	29
BCC2 lightweight, integer	n/a	14	14	n/a	14	14
BCC2 lightweight, floating-point	n/a	14	14	n/a	14	14
BCC2 heavyweight, integer	n/a	29	29	n/a	29	29
BCC2 heavyweight, floating-point	n/a	29	29	n/a	29	29
ECC1 heavyweight, integer	n/a	n/a	n/a	31	31	31
ECC1 heavyweight, floating-point	n/a	n/a	n/a	31	31	31
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	31
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	31

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	22	22	23	22	22	23
BCC1 lightweight, floating-point	23	23	24	23	23	24
BCC1 heavyweight, integer	40	40	41	40	40	41
BCC1 heavyweight, floating-point	40	40	41	40	40	41
BCC2 lightweight, integer	n/a	23	24	n/a	23	24
BCC2 lightweight, floating-point	n/a	23	24	n/a	23	24
BCC2 heavyweight, integer	n/a	40	41	n/a	40	41
BCC2 heavyweight, floating-point	n/a	40	41	n/a	40	41
ECC1 heavyweight, integer	n/a	n/a	n/a	41	41	42
ECC1 heavyweight, floating-point	n/a	n/a	n/a	41	41	42
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	42
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	42
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	31	31	31	31	31	31
BCC1 lightweight, floating-point	32	32	32	32	32	32
BCC1 heavyweight, integer	49	49	49	49	49	49
BCC1 heavyweight, floating-point	49	49	49	49	49	49
BCC2 lightweight, integer	n/a	32	32	n/a	32	32
BCC2 lightweight, floating-point	n/a	32	32	n/a	32	32
BCC2 heavyweight, integer	n/a	49	49	n/a	49	49
BCC2 heavyweight, floating-point	n/a	49	49	n/a	49	49
ECC1 heavyweight, integer	n/a	n/a	n/a	50	50	50
ECC1 heavyweight, floating-point	n/a	n/a	n/a	50	50	50
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	50
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	50

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	22	22	23	22	22	23
BCC1 lightweight, floating-point	23	23	24	23	23	24
BCC1 heavyweight, integer	40	40	41	40	40	41
BCC1 heavyweight, floating-point	40	40	41	40	40	41
BCC2 lightweight, integer	n/a	23	24	n/a	23	24
BCC2 lightweight, floating-point	n/a	23	24	n/a	23	24
BCC2 heavyweight, integer	n/a	40	41	n/a	40	41
BCC2 heavyweight, floating-point	n/a	40	41	n/a	40	41
ECC1 heavyweight, integer	n/a	n/a	n/a	49	49	50
ECC1 heavyweight, floating-point	n/a	n/a	n/a	49	49	50
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	50
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	50
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	31	31	31	31	31	31
BCC1 lightweight, floating-point	32	32	32	32	32	32
BCC1 heavyweight, integer	49	49	49	49	49	49
BCC1 heavyweight, floating-point	49	49	49	49	49	49
BCC2 lightweight, integer	n/a	32	32	n/a	32	32
BCC2 lightweight, floating-point	n/a	32	32	n/a	32	32
BCC2 heavyweight, integer	n/a	49	49	n/a	49	49
BCC2 heavyweight, floating-point	n/a	49	49	n/a	49	49
ECC1 heavyweight, integer	n/a	n/a	n/a	58	58	58
ECC1 heavyweight, floating-point	n/a	n/a	n/a	58	58	58
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	58
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	58

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for the TriCore17x6/Tasking 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 are 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 substitutions in the following table should be used:

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

Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found on the ETAS Group website www.etasgroup.com.