RTA-OSEK

Binding Manual: HC12X/Metrowerks

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

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

www.etas.de

Japan

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

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

www.etas.co.jp

Korea

ETAS Korea Co. Ltd.
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

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

France

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

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

www.etas.fr

Great Britain

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

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

www.etas-uk.net

Copyright Notice

Version: RM00070-002

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 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 5

1.1 Who Should Read this Guide? 5

1.2 Conventions 5

2 Toolchain Issues 7

2.1 Memory Model 7

2.1.1 Background 7

2.1.2 Function Addresses 8

2.1.3 Data Addresses 8

2.1.4 Preservation of Page Registers by ISRs 8

2.2 Compiler 9

2.3 Assembler 10

2.4 Linker/Locator 10

2.4.1 Compiler Sections 11

2.4.2 Section Placement 11

2.4.3 Pointers Passed to RTA-OSEK API Functions 12

2.5 Debugger....13

3 Target Hardware Issues....15

3.1 Interrupts....15

3.1.1 Interrupt Levels....15

3.1.2 Interrupt Vectors....15

3.1.3 Category 1 Handlers....16

3.1.4 Category 2 Handlers....16

3.1.5 Vector Table Issues....16

3.1.6 Interrupt Priority Levels....17

3.2 Register Settings....17

3.3 Stack Usage....18

3.3.1 Number of Stacks....18

3.3.2 Stack Usage within API Calls....18

4 Parameters of Implementation....19

4.1 Functionality....19

4.2 Hardware Resources....20

4.2.1 ROM and RAM Overheads....20

4.2.2 ROM and RAM for OSEK OS Objects....21

4.2.3 Size of Linkable Modules....26

4.2.4 Reserved Hardware Resources....39

4.3 Performance....39

4.3.1 Execution Times for RTA-OSEK API Calls....39

4.3.2 OS Start-up Time....48

4.3.3 Interrupt Latencies....49

4.3.4 Task Switching Times....50

4.4 Configuration of Run-time Context....53

1 About this Guide

This guide provides port specific information for the HC12X/Metrowerks implementation of LiveDevices' RTA-OSEK.

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

1.1 Who Should Read this Guide?

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

1.2 Conventions

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

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

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

2 Toolchain Issues

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

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

2.1 Memory Model

The S12X architecture offers three memory models: small, banked, and large. **This port is built for the large model.**

2.1.1 Background

The S12X has a 16-bit logical address space. Standard S12X instructions always use logical addresses. Within this logical address space there are 3 banked memory-windows: one for Flash memory at addresses `0x8000` to `0xBFFF`, one for RAM at addresses `0x1000` to `0x1FFF` and one for EEPROM at addresses `0x0800` to `0x0BFF`. Whenever a standard S12X instruction references memory in the Flash banked memory-window the contents of the `PPAGE` register are used to determine which of several Flash memory pages is currently visible in the memory-window. Likewise the `RPAGE` and `EPAGE` registers are used for the RAM and EEPROM banked memory-windows respectively.

Placement of code and data into memory pages is controlled by the Metrowerks linker file. See the "Segmentation" section of the "Freescale HC12 Back End" chapter of the Metrowerks Compiler Manual for details.

In addition to its 16-bit logical address space the S12X also has a 23-bit global address space. This global address space contains all physical memory. That is, all of the Flash, RAM and EEPROM pages as well as the unbanked memory are simultaneously visible in the global address space. The most significant 7 bits of a global address are taken from the `GPAGE` register and the least significant 16 bits come from the address field of an extended `Gxxxx` instruction (e.g. `GLDAA` or `GSTAA`). The Metrowerks linker uses information in the linker file to generate 23 bit global addresses.

There is a good description of how memory works in the S12X in the Metrowerks library source file `<install>\lib\hc12c\src\DATAPAGE.C`. Where `<install>` is the root of the Metrowerks toolchain installation.

Note: when this manual refers to "unbanked ROM" is means Flash memory **not** in the `0x8000` to `0xBFFF` banked memory-window. Likewise when this manual refers to "unbanked RAM" it means RAM **not** in the `0x1000` to `0x1FFF` banked memory-window.

2.1.2 Function Addresses

In the large memory model functions have 3-byte "far" addresses by default. The most significant byte indicates the page of Flash that should be mapped into the banked memory-window between `0x8000` and `0xBFFF`. The currently-mapped page is stored in the `PPAGE` register and the assembly instructions `CALL` and `RTC` maintain this implicitly.

"Near" functions have only 2-byte addresses, and must be present in the 16-bit logical address space at the time of calling. Near functions must be placed in an unbanked memory-area.

2.1.3 Data Addresses

In the large memory model, data is "far" by default, with a 3-byte address. Far data pointers are 3 bytes but have the page register in the least significant byte - unlike function pointers. The page register part of a far data pointer is written to the `GPAGE` register and the compiler generates extended `Gxxx` instructions to access the far memory. The combination of the `GPAGE` register and the extended `Gxxx` instructions allow a far data reference to access any location in physical memory (i.e. all unbanked memory and all RAM, EEPROM and flash pages).

"Near" data only has a 2-byte address and consequently pointers to near data are 2 bytes. Near data must be placed in an unbanked memory area.

2.1.4 Preservation of Page Registers by ISRs

Note: for Category 2 ISRs RTA-OSEK automatically preserves the `PPAGE`, `GPAGE`, `RPAGE` and `EPAGE` page registers. For Category 1 ISRs the user must pass the appropriate `–CpGPAGE`, `-CpRPAGE` and `–CpEPAGE` options to the compiler so that `__interrupt` functions preserve these page registers. See the Metrowerks Compiler Manual for more details.

2.2 Compiler

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

Vendor	Metrowerks
Compiler	CHC12.EXE
Version	5.0.28 Build 5051

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

Option	**Description**
`-Ml`	Build for the large memory model
`-CpPPAGE=0x30`	Tell the compiler that the PPAGE register is located at address 0x30
`-CpGPAGE`	Tell the compiler that the GPAGE register is used

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**
`-Ml`	Build for large memory model
`-CpuHCS12X`	Use extended instruction set. Without this, RTA-OSEK's stack figures may be incorrect.
`-CpPPAGE=0x30`	Tell the compiler that the PPAGE register is located at address 0x30
`-CpGPAGE`	Tell the compiler that the GPAGE register is used

Note: When compiling the automatically generated file `osekdefs.c` you may see warnings saying that functions defined in `osekdefs.c` were previously declared in different segments. This is an artifact of the way that `osekdefs.c` is generated and the warnings may be ignored.

Note: When compiling the automatically generated RTA-TRACE file `ostrace.c` you may see warnings saying that variables defined in `ostrace.c` were previously declared in different segments. This is an artifact of the way that `ostrace.c` is generated and the warnings may be ignored.

2.3 Assembler

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

Vendor	Metrowerks
Assembler	AHC12.EXE
Version	5.0.28 build 5051

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

Option	Description
`-Ml`	Build for the large memory model

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

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

Option	Description
`-Ml`	Build for the large memory model
`-CpuHCS12X`	Enable extended instruction set

2.4 Linker/Locator

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

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

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

C Library Functions	Description
`_FPCMP`	Far pointer comparison

This port of RTA-OSEK is built for the ansixl.lib, ansixli.lib or ansixlf.lib runtime libraries. The compiler flag –Ml specifies the large memory model. It is a linker error to attempt to mix modules built for different memory models.

2.4.1 Compiler Sections

RTA-OSEK code and data are not placed in the default compiler generated sections. The sections used by RTA-OSEK are as follows:

RTA-OSEK Section	Section Contents
os_text	Code – located in banked or unbanked memory.
os_text_unbanked	Interrupt handling code – must be located in unbanked memory.
os_pid	Read-only data – must be located in unbanked memory.
os_pird	Read-only data – must be located in unbanked memory.
os_constdata_unbanked	Read-only data – must be located in unbanked memory.
os_pur	Writeable data – must be located in unbanked RAM.
os_pir	Writeable data – must be located in unbanked RAM.
os_data_unbanked	Writeable data – must be located in unbanked RAM.
os_intvec	Vector table – must be located in unbanked memory.
os_intvechi	High vector table – must be located in unbanked memory.

2.4.2 Section Placement

The vector table must be located in unbanked memory. If the vector table is generated by RTA-OSEK then it is in sections os_intvec and os_intvechi, which must be located in unbanked memory with the PLACEMENT clause of the linker file.

Some of the RTA-OSEK code concerned with interrupt handling must be loaded into unbanked memory. This code is in section os_text_unbanked, which should appear in the `PLACEMENT` section of the linker file and be mapped to unbanked memory. E.g. the unbanked flash areas `0x4000..0x7FFF` or `0xC000..0xFF0F`.

All other RTA-OSEK code is in the `os_text` section. This section may be placed in banked or unbanked memory.

The kernel expects to find its internal read-only variables in near memory. Read-only variables are in the `os_pid`, `os_pird` and `os_constdata_unbanked` sections, which should appear in the `PLACEMENT` section of the linker file and be mapped to unbanked memory. E.g. the unbanked flash areas `0x4000..0x7FFF` or `0xC000..0xFF0F`.

The kernel expects to find its writeable internal variables in near memory. Writeable variables are in the `os_pir`, `os_pur` and `os_data_unbanked` sections, which should appear in the `PLACEMENT` section of the linker file and be mapped to the unbanked RAM area `0x2000..0x3FFF`.

The linker will omit code that is not referenced by other code unless it appears in the `ENTRIES` section in the linker file. Thus, `os_intvec` and `os_intvechi` must appear in the `ENTRIES` section if used.

2.4.3 Pointers Passed to RTA-OSEK API Functions

Note: All RTA-OSEK API pointer arguments are near (i.e. they are decorated with the `__near` modifier). This means that if you call an RTA-OSEK API function that has a pointer argument, the object to which the pointer points must be in near memory. For example if you call the RTA-OSEK API function `GetAlarmBase(AlarmType, AlarmBaseRefType)` then the `AlarmBaseType` object pointed to by the `AlarmBaseRefType` type argument must be in near memory. Objects allocated on the stack are always in near memory. To place a global object in near memory use `#pragma DATA_SEG` or `#pragma CONST_SEG` with a `__NEAR_SEG` modifier. See the "Segmentation" section of the "Freescale HC12 Back End" chapter of the Metrowerks Compiler Manual. The section declared with the pragma must then be placed in unbanked memory by the linker file.

Any objects created by RTA-OSEK will be automatically placed in near memory. To ensure that more efficient near accesses (i.e. accesses that do not need to load the `GPAGE` register) are made to objects created by RTA-OSEK the automatically generated header files included by application source code (e.g. `osek.h` and `oseklib.h`) start with the code:

```
#pragma push
#pragma CONST_SEG __NEAR_SEG os_constdata_unbanked
#pragma DATA_SEG __NEAR_SEG os_data_unbanked
```

and end with the code:

```
#pragma pop
```

This code informs the compiler that all objects declared in the header file (or any nested header files) are in the near sections `os_constdata_unbanked` or `os_data_unbanked` and can be accessed with near memory references.

As a result of the above, if you re-declare an object that is declared in one of the automatically generated header files (e.g. using `DeclareTask()`) you may get a compiler warning saying that the object was previously declared in a different segment. These warnings are benign. Application code will generate far references to the near data. This is inefficient, but works. To avoid the warnings and inefficiencies there are two options.

- Do not re-declare objects declared inside the header files automatically generated by RTA-OSEK.
- If you do re-declare objects, use code like the following :

```
#pragma push
#pragma CONST_SEG __NEAR_SEG os_constdata_unbanked
#pragma DATA_SEG __NEAR_SEG os_data_unbanked

/* Re-declaration. */

#pragma pop
```

2.5 Debugger

Information about ORTI for RTA-OSEK can be found in the *RTA-OSEK ORTI Guide*

This port does not include support for any specific ORTI aware debuggers.

If you are using an ORTI version 2.0 aware debugger on this platform you can use the "Unknown ORTI debugger" option in the RTA-OSEK GUI to generate an ORTI output file. The ORTI generated will not have been tested on the debugger and, therefore, is not guaranteed to work.

Please contact LiveDevices if you have any questions about ORTI support in RTA-OSEK.

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 *MC9S12XDP512 Device User Guide*.

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

IPL Value	CCRW	Description
0	CCR.I = 0; CCRH = 0	User level
1..7	CCR.I = 0; CCRH = [1..7]	Category 1 and 2 interrupts
8	CCR.I = 1; CCRH = any	Non-maskable interrupts only

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
Fixed vectors 0xFFFA, 0xFFFC	Non-maskable interrupts: must be defined as Category 1 and have IPL of 8
Relocatable vectors 0xF4 to 0xF8	Non-maskable interrupts: must be defined as Category 1 and have IPL of 8
Relocatable vectors 0x60 to 0xF3	Priority 1..7; may be defined as category 1 or 2.
Relocatable vector 0x10	Non-maskable interrupt: must be defined as Category 1 and have IPL of 8

The valid base addresses for the vector table are:

Base Register	Notes
0xFF	Default base address
0x40..0x7F, 0xC0..0xFE	Unpaged Flash
0x80..0xBF	Paged Flash. Only valid if application does not page.

Base Register	Notes
`0x08..0x0B, 0x10..0x3F`	Paged EEPROM; paged and unpaged SRAM. Not recommended. User is responsible for setup and for managing side-effects.
`0x0C..0x0F`	Unpaged EEPROM. Not recommended.

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 Metrowerks C compiler can generate appropriate interrupt handling code for a C function decorated with the `__interrupt` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

When you configure your application with the RTA-OSEK GUI you can choose whether or not a vector table is generated within `osgen.asm`.

Note that a generated vector table omits the reset vector entry. If you choose to provide your own vector table, it must contain an entry for each interrupt handler, including the Category 2 interrupt handlers in RTA-OSEK.

The following table shows the syntax for labels attached to RTA-OSEK Category 2 interrupt handlers (xx represents the 2 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
`IVBR` + *xx*	os_wrapper_*xx*

The S12X has some vectors that have a fixed location (vectors 0xFFFC, 0xFFFA) and the rest of the vectors are relocatable. If a vector table is generated it is in two sections. Section `os_intvechi` contains the fixed location vectors and must be located at 0xFFFA. Section `os_intvec` contains the relocatable vectors and can be placed at any of the locations outlined in the processor documentation. The user is responsible for initializing the Interrupt Vector Base Register (IVBR).

3.1.6 Interrupt Priority Levels

The priority at which a hardware interrupt is taken is set in the `INT_CFDATA` registers under the control of the `INT_CFADDR` register.

The RTA-OSEK GUI generates a table, called `os_InitIrqLevels`, which must be used to initialize the `INT_CFDATA` registers. This table contains the priority levels for interrupts defined in the application.

Important: The `os_InitIrqLevels` table must be copied to the `INT_CFDATA` registers before the call to `StartOS()` otherwise interrupts will not work correctly.

The `init_target()` function in `target.c` in the example application, located in *<RTA-OSEK install directory>*`\HC12XMW\Example\`, gives an example of how to copy `os_InitIrqLevels` to the correct location.

3.2 Register Settings

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

Register	Value	Notes
`CCRH`	os_oim	Set the IPL to block out all category 2 interrupts
`CCR.I`	0	clear the I bit in CCR (e.g. by issuing `__asm ( cli );`)
`INT_CFDATA`x	*y*	Initialize interrupt priority table

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

Registers	Notes
`CCRH`	After startOS this must not be altered
`CCR.I`	the only valid action for this bit is to clear it on entry to an ISR at one of the non-maskable vectors.

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 short`.

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): 20

Timing

API max usage (bytes): 20

Extended

API max usage (bytes): 30

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.

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	**9S12XDP512MPV**
Clock speed (MHz)	8
Code memory	Internal Flash
Read-only data memory	Internal Flash
Read-write data memory	Internal RAM

4.1 Functionality

The OSEK Operating System Specification specifies four conformance classes. These attributes apply to *systems* built with OSEK OS objects. The following table specifies the number of OSEK OS and COM objects supported per conformance class.

Configuration	**Application Uses**					
Events	**No**			**Yes**		
Shared Task Priorities	**No**		**Yes**	**No**		**Yes**
Multiple Task Activations	**No**	**Yes**		**No**	**Yes**	
Maximum number of tasks	16	16	16	16	16	16
Maximum number of not suspended tasks	16	16	16	16	16	16
Maximum number of priorities	16	16	16	16	16	16
Number of tasks per priority (for BCC2 and ECC2)	n/a	16	16	n/a	16	16
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	16	16	16
Limits for the number of alarm objects (per system / per task)	not limited by RTA-OSEK					
Limits for the number of standard resources (per system)	255	255	255	255	255	255
Limits for the number of internal resources (per system)	not limited by RTA-OSEK					
Limits for the number of nested resources (per system / per task)	255	255	255	255	255	255

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Limits for the number of application modes	255					

4.2 Hardware Resources

4.2.1 ROM and RAM Overheads

The following tables give the ROM and RAM overheads for the RTA-OSEK Component (in bytes). The OSEK COM overheads are quoted separately. If you do not use messages, your application will not include this overhead for the parts of OSEK COM required to implement messaging.

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	12	12	12	12	12	12
	ROM	95	95	95	95	95	95
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	22	22	22	22	22	22
	ROM	130	130	130	130	130	130
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	28	28	28	28	28	28
	ROM	150	150	150	150	150	150
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

4.2.2 ROM and RAM for OSEK OS Objects

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

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

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
BCC1 Heavyweight task	RAM	2	2	2	2	2	2
	ROM	22	22	22	22	22	22
BCC2 task	RAM	n/a	3	5	n/a	3	5
	ROM	n/a	23	27	n/a	23	27
ECC1, Integer task	RAM	n/a	n/a	n/a	11	11	11
	ROM	n/a	n/a	n/a	32	32	32
ECC1, floating-point task	RAM	n/a	n/a	n/a	12	12	12
	ROM	n/a	n/a	n/a	32	32	32
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	13
	ROM	n/a	n/a	n/a	n/a	n/a	36
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	14
	ROM	n/a	n/a	n/a	n/a	n/a	36
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	38	38	38	38	38	38
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	51	51	51	51	51	51
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Alarm	RAM	5	5	5	5	5	5
	ROM	36	36	36	36	36	36
Counter	RAM	2	2	2	2	2	2
	ROM	4	4	4	4	4	4
Message	RAM	11	11	11	31	31	31
	ROM	11	11	11	28	28	28
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	7	7	7	7	7	7
	ROM	20	20	20	20	20	20
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	5	5	5	5	5	5
	ROM	25	25	25	25	25	25
BCC1 Heavyweight task	RAM	7	7	7	7	7	7
	ROM	27	27	27	27	27	27
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	28	32	n/a	28	32
ECC1, Integer task	RAM	n/a	n/a	n/a	16	16	16
	ROM	n/a	n/a	n/a	37	37	37
ECC1, floating-point task	RAM	n/a	n/a	n/a	17	17	17
	ROM	n/a	n/a	n/a	37	37	37
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	18
	ROM	n/a	n/a	n/a	n/a	n/a	41
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	19
	ROM	n/a	n/a	n/a	n/a	n/a	41

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	5	5	5	5	5	5
	ROM	59	59	59	59	59	59
Category 2 ISR, floating-point	RAM	6	6	6	6	6	6
	ROM	67	67	67	67	67	67
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Alarm	RAM	5	5	5	5	5	5
	ROM	36	36	36	36	36	36
Counter	RAM	2	2	2	2	2	2
	ROM	4	4	4	4	4	4
Message	RAM	11	11	11	31	31	31
	ROM	11	11	11	28	28	28
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	7	7	7	7	7	7
	ROM	20	20	20	20	20	20
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	6	6	6	6	6	6
	ROM	29	29	29	29	29	29
BCC1 Heavyweight task	RAM	8	8	8	8	8	8
	ROM	29	29	29	29	29	29
BCC2 task	RAM	n/a	9	11	n/a	9	11
	ROM	n/a	30	34	n/a	30	34
ECC1, Integer task	RAM	n/a	n/a	n/a	17	17	17
	ROM	n/a	n/a	n/a	39	39	39
ECC1, floating-point task	RAM	n/a	n/a	n/a	18	18	18
	ROM	n/a	n/a	n/a	39	39	39
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	19
	ROM	n/a	n/a	n/a	n/a	n/a	43
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	20
	ROM	n/a	n/a	n/a	n/a	n/a	43
Category 2 ISR	RAM	6	6	6	6	6	6
	ROM	63	63	63	63	63	63
Category 2 ISR, floating-point	RAM	7	7	7	7	7	7
	ROM	71	71	71	71	71	71
Resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Alarm	RAM	5	5	5	5	5	5
	ROM	38	38	38	38	38	38
Counter	RAM	2	2	2	2	2	2
	ROM	6	6	6	6	6	6
Message	RAM	11	11	11	31	31	31
	ROM	13	13	13	30	30	30
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (writable)	RAM	10	10	10	10	10	10
	ROM	10	10	10	10	10	10
Schedule	RAM	9	9	9	9	9	9
	ROM	24	24	24	24	24	24
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

4.2.3 Size of Linkable Modules

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

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

The call variants are as follows:

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

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	85	141	179	89	145	205
	NS		69	127	159	73	131	185
	KL	2	41	109	145	45	113	167

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	18	18	18	18	18	18
ChainTask	SWL	1, 8	75	124	161	79	128	183
	SWH	1, 9	100	148	185	104	152	207
	NSL	8	75	124	161	79	128	183
	NSH	9	94	142	179	98	146	201
Schedule			51	51	74	51	51	74
GetTaskID			21	21	21	21	21	21
GetTaskState			61	61	61	77	77	77
EnableAllInterrupts			7	7	7	7	7	7
DisableAllInterrupts			14	14	14	14	14	14
ResumeAllInterrupts			15	15	15	15	15	15
SuspendAllInterrupts			22	22	22	22	22	22
ResumeOSInterrupts			15	15	15	15	15	15
SuspendOSInterrupts			32	32	32	32	32	32
GetResource	Task	7	22	22	26	22	22	26
	Combined	6	62	62	62	62	62	62
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	42	42	42	42	42	42
	Combined	6	86	86	86	86	86	86
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	90	90	185
	NS		n/a	n/a	n/a	76	76	150
	NS1i	10	n/a	n/a	n/a	32	n/a	n/a
	KL	2	n/a	n/a	n/a	52	52	132
	KL1i	2, 10	n/a	n/a	n/a	13	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	176	176	345
	fp	11	n/a	n/a	n/a	200	200	396
	1i	10	n/a	n/a	n/a	21	n/a	n/a
GetAlarmBase			37	37	37	37	37	37
GetAlarm			69	69	69	69	69	69
SetRelAlarm			98	98	98	98	98	98
SetAbsAlarm			112	112	112	112	112	112
CancelAlarm			53	53	53	53	53	53
InitCounter			56	56	56	56	56	56

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			49	49	49	49	49	49
osek_tick_alarm	<default>		54	54	54	54	54	54
	KL	2	29	29	29	29	29	29
osek_incr_counter			27	27	27	27	27	27
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			88	88	88	88	88	88
ShutdownOS	NoHook	12	16	16	16	16	16	16
	Hook	13	24	24	24	24	24	24
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			14	14	14	14	14	14
StopCOM			9	9	9	9	9	9
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	48	48	48	136	136	136
	CCCB	15	136	136	136	136	136	136
GetMessageResource			21	21	21	21	21	21
ReleaseMessageResource			19	19	19	19	19	19
GetMessageStatus			37	37	37	37	37	37
SendMessage	SW CCCA	1, 14	79	79	79	199	199	199
	SW CCCB	1, 15	183	183	183	199	199	199
	NS CCCA	14	79	79	79	199	199	199
	NS CCCB	15	183	183	183	199	199	199
	KL CCCA	2, 14	50	50	50	177	177	177
	KL CCCB	2, 15	163	163	163	177	177	177
main_dispatch	NoHook	12	82	82	113	82	82	113
	Hook	13	108	108	139	108	108	139
sub_dispatch	B1LF	19	17	17	17	17	17	17
	B1HI	20	57	57	57	57	57	57
	B1HF	21	65	65	65	65	65	65
	B2LI	22	n/a	41	70	n/a	41	70
	B2LF	23	n/a	49	78	n/a	49	78
	B2HI	24	n/a	94	158	n/a	94	158
	B2HF	25	n/a	102	166	n/a	102	166
	E1HI	26	n/a	n/a	n/a	249	249	316
	E1HF	27	n/a	n/a	n/a	257	257	324
	E2HI	28	n/a	n/a	n/a	n/a	n/a	316

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	324
ErrorHook support		16	18	18	18	18	18	18
	ServiceID	17	23	23	23	23	23	23
	Parameters	18	38	38	38	38	38	38
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	42	95	148	48	110	178
	NS		30	82	135	36	97	165
	KL	2	13	65	119	19	81	149
ChainTaskset	SWL	1, 8	31	87	138	31	97	162
	SWH	1, 9	68	118	167	68	128	179
	NSL	8	31	87	138	31	97	162
	NSH	9	62	112	161	62	122	173
GetTasksetRef			10	10	10	10	10	10
MergeTaskset			32	32	32	32	32	32
AssignTaskset			10	10	10	10	10	10
RemoveTaskset			32	32	32	32	32	32
TestSubTaskset			48	48	48	48	48	48
TestEquivalentTaskset			42	42	42	42	42	42
TickSchedule	SW	1	104	103	103	103	103	103
	NS		87	86	86	86	86	86
	KL	2	69	68	68	68	68	68
AdvanceSchedule	SW	1	105	102	102	102	102	102
	NS		86	83	83	83	83	83
	KL	2	70	67	67	67	67	67
StartSchedule			55	55	55	55	55	55
StopSchedule			39	39	39	39	39	39
GetScheduleStatus			68	68	68	68	68	68
GetScheduleValue			47	47	47	47	47	47
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			10	10	10	10	10	10
SetArrivalpointDelay			6	6	6	6	6	6
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			6	6	6	6	6	6

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestArrivalpointWritable			22	22	22	22	22	22
GetExecutionTime			3	3	3	3	3	3
GetLargestExecutionTime			7	7	7	7	7	7
ResetLargestExecutionTime			2	2	2	2	2	2
GetStackOffset			18	18	18	18	18	18

Timing

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	85	141	179	89	145	205
	NS		69	127	159	73	131	185
	KL	2	41	109	145	45	113	167
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	75	124	161	79	128	183
	SWH	1, 9	100	148	185	104	152	207
	NSL	8	75	124	161	79	128	183
	NSH	9	94	142	179	98	146	201
Schedule			60	60	83	60	60	83
GetTaskID			21	21	21	21	21	21
GetTaskState			61	61	61	77	77	77
EnableAllInterrupts			7	7	7	7	7	7
DisableAllInterrupts			14	14	14	14	14	14
ResumeAllInterrupts			15	15	15	15	15	15
SuspendAllInterrupts			22	22	22	22	22	22
ResumeOSInterrupts			15	15	15	15	15	15
SuspendOSInterrupts			32	32	32	32	32	32
GetResource	Task	7	22	22	26	22	22	26
	Combined	6	62	62	62	62	62	62
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	51	51	51	51	51	51
	Combined	6	104	104	104	104	104	104

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	90	90	185
	NS		n/a	n/a	n/a	76	76	150
	NS1i	10	n/a	n/a	n/a	32	n/a	n/a
	KL	2	n/a	n/a	n/a	52	52	132
	KL1i	2, 10	n/a	n/a	n/a	13	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	211	211	380
	fp	11	n/a	n/a	n/a	235	235	431
	1i	10	n/a	n/a	n/a	51	n/a	n/a
GetAlarmBase			37	37	37	37	37	37
GetAlarm			69	69	69	69	69	69
SetRelAlarm			98	98	98	98	98	98
SetAbsAlarm			112	112	112	112	112	112
CancelAlarm			53	53	53	53	53	53
InitCounter			56	56	56	56	56	56
GetCounterValue			49	49	49	49	49	49
osek_tick_alarm	<default>		54	54	54	54	54	54
	KL	2	29	29	29	29	29	29
osek_incr_counter			27	27	27	27	27	27
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			115	115	115	115	115	115
ShutdownOS	NoHook	12	16	16	16	16	16	16
	Hook	13	24	24	24	24	24	24
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			14	14	14	14	14	14
StopCOM			9	9	9	9	9	9
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	48	48	48	136	136	136
	CCCB	15	136	136	136	136	136	136
GetMessageResource			21	21	21	21	21	21
ReleaseMessageResource			19	19	19	19	19	19
GetMessageStatus			37	37	37	37	37	37
SendMessage	SW CCCA	1, 14	79	79	79	199	199	199

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
	SW CCCB	1, 15	183	183	183	199	199	199
	NS CCCA	14	79	79	79	199	199	199
	NS CCCB	15	183	183	183	199	199	199
	KL CCCA	2, 14	50	50	50	177	177	177
	KL CCCB	2, 15	163	163	163	177	177	177
main_dispatch	NoHook	12	127	127	167	127	127	167
	Hook	13	160	160	198	160	160	198
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	63	63	63	63	63	63
	B1HF	21	73	73	73	73	73	73
	B2LI	22	n/a	31	60	n/a	31	60
	B2LF	23	n/a	39	68	n/a	39	68
	B2HI	24	n/a	88	152	n/a	88	152
	B2HF	25	n/a	96	160	n/a	96	160
	E1HI	26	n/a	n/a	n/a	262	262	329
	E1HF	27	n/a	n/a	n/a	270	270	337
	E2HI	28	n/a	n/a	n/a	n/a	n/a	329
	E2HF	29	n/a	n/a	n/a	n/a	n/a	337
ErrorHook support		16	18	18	18	18	18	18
	ServiceID	17	23	23	23	23	23	23
	Parameters	18	38	38	38	38	38	38
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	38	38	38	38	38	38
Timing_termination		4	53	53	53	53	53	53
ActivateTaskset	SW	1	42	95	148	48	110	178
	NS		30	82	135	36	97	165
	KL	2	13	65	119	19	81	149
ChainTaskset	SWL	1, 8	31	87	138	31	97	162
	SWH	1, 9	68	118	167	68	128	179
	NSL	8	31	87	138	31	97	162
	NSH	9	62	112	161	62	122	173
GetTasksetRef			10	10	10	10	10	10
MergeTaskset			32	32	32	32	32	32
AssignTaskset			10	10	10	10	10	10
RemoveTaskset			32	32	32	32	32	32
TestSubTaskset			48	48	48	48	48	48
TestEquivalentTaskset			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	
TickSchedule	SW	1	104	103	103	103	103	103
	NS		87	86	86	86	86	86
	KL	2	69	68	68	68	68	68
AdvanceSchedule	SW	1	105	102	102	102	102	102
	NS		86	83	83	83	83	83
	KL	2	70	67	67	67	67	67
StartSchedule			55	55	55	55	55	55
StopSchedule			39	39	39	39	39	39
GetScheduleStatus			68	68	68	68	68	68
GetScheduleValue			47	47	47	47	47	47
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			10	10	10	10	10	10
SetArrivalpointDelay			6	6	6	6	6	6
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			6	6	6	6	6	6
TestArrivalpointWritable			22	22	22	22	22	22
GetExecutionTime			47	47	47	47	47	47
GetLargestExecutionTime			14	14	14	14	14	14
ResetLargestExecutionTime			11	11	11	11	11	11
GetStackOffset			18	18	18	18	18	18

Extended

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	177	232	272	181	237	296
	NS		209	265	302	213	269	326
	KL	2	118	178	216	122	182	240
TerminateTask	LExt	3	73	73	73	73	73	73
	H	5	97	97	97	97	97	97
ChainTask	SWL	1, 8	192	252	289	196	256	315

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	SWH	1, 9	220	268	306	224	272	326
	NSL	8	245	307	344	251	311	375
	NSH	9	265	318	356	271	322	379
Schedule			145	145	166	145	145	166
GetTaskID			31	31	31	31	31	31
GetTaskState			167	167	167	171	171	171
EnableAllInterrupts			17	17	17	17	17	17
DisableAllInterrupts			24	24	24	24	24	24
ResumeAllInterrupts			48	48	48	48	48	48
SuspendAllInterrupts			32	32	32	32	32	32
ResumeOSInterrupts			48	48	48	48	48	48
SuspendOSInterrupts			42	42	42	42	42	42
GetResource	Task	7	291	291	240	291	291	240
	Combined	6	274	274	274	274	274	274
	CLEx	3	240	240	240	240	240	240
ReleaseResource	Task	7	242	242	242	242	242	242
	Combined	6	270	270	270	270	270	270
	CLEx	3	234	234	234	234	234	234
SetEvent	SW	1	n/a	n/a	n/a	208	208	304
	NS		n/a	n/a	n/a	258	258	333
	NS1i	10	n/a	n/a	n/a	151	n/a	n/a
	KL	2	n/a	n/a	n/a	168	168	238
	KL1i	2, 10	n/a	n/a	n/a	126	n/a	n/a
ClearEvent			n/a	n/a	n/a	89	89	89
GetEvent			n/a	n/a	n/a	124	124	124
WaitEvent	<default>		n/a	n/a	n/a	295	295	462
	fp	11	n/a	n/a	n/a	319	319	513
	1i	10	n/a	n/a	n/a	135	n/a	n/a
GetAlarmBase			130	130	130	130	130	130
GetAlarm			125	125	125	125	125	125
SetRelAlarm			177	177	177	177	177	177
SetAbsAlarm			189	189	189	189	189	189
CancelAlarm			121	121	121	121	121	121
InitCounter			181	181	181	181	181	181
GetCounterValue			132	132	132	132	132	132
osek_tick_alarm	<default>		83	83	83	83	83	83
	KL	2	29	29	29	29	29	29

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
osek_incr_counter			27	27	27	27	27	27
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			125	125	125	125	125	125
ShutdownOS	NoHook	12	21	21	21	21	21	21
	Hook	13	29	29	29	29	29	29
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			24	24	24	24	24	24
StopCOM			24	24	24	24	24	24
ReadFlag			18	18	18	18	18	18
ResetFlag			19	19	19	19	19	19
ReceiveMessage	CCCA	14	120	120	120	202	202	202
	CCCB	15	202	202	202	202	202	202
GetMessageResource			60	60	60	60	60	60
ReleaseMessageResource			60	60	60	60	60	60
GetMessageStatus			68	68	68	68	68	68
SendMessage	SW CCCA	1, 14	156	156	156	260	260	260
	SW CCCB	1, 15	246	246	246	260	260	260
	NS CCCA	14	156	156	156	260	260	260
	NS CCCB	15	246	246	246	260	260	260
	KL CCCA	2, 14	111	111	111	234	234	234
	KL CCCB	2, 15	218	218	218	234	234	234
main_dispatch	NoHook	12	127	127	167	127	127	167
	Hook	13	160	160	198	160	160	198
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	63	63	63	63	63	63
	B1HF	21	73	73	73	73	73	73
	B2LI	22	n/a	31	60	n/a	31	60
	B2LF	23	n/a	39	68	n/a	39	68
	B2HI	24	n/a	88	152	n/a	88	152
	B2HF	25	n/a	96	160	n/a	96	160
	E1HI	26	n/a	n/a	n/a	262	262	334
	E1HF	27	n/a	n/a	n/a	270	270	342
	E2HI	28	n/a	n/a	n/a	n/a	n/a	334
	E2HF	29	n/a	n/a	n/a	n/a	n/a	342
ErrorHook support		16	71	71	71	71	71	71
	ServiceID	17	76	76	76	76	76	76

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Parameters	18	91	91	91	91	91	91
validity_checks		3	43	43	43	43	43	43
Timing_dispatch		4	38	38	38	38	38	38
Timing_termination		4	53	53	53	53	53	53
ActivateTaskset	SW	1	209	259	308	219	282	343
	NS		249	298	347	259	321	382
	KL	2	165	215	262	175	238	298
ChainTaskset	SWL	1, 8	250	311	358	260	326	385
	SWH	1, 9	295	364	416	304	379	440
	NSL	8	301	360	409	310	375	436
	NSH	9	340	407	459	349	422	483
GetTasksetRef			93	93	93	93	93	93
MergeTaskset			169	169	169	169	169	169
AssignTaskset			114	114	114	114	114	114
RemoveTaskset			167	167	167	167	167	167
TestSubTaskset			171	171	171	171	171	171
TestEquivalentTaskset			165	165	165	165	165	165
TickSchedule	SW	1	257	192	192	192	192	192
	NS		335	258	258	258	258	258
	KL	2	211	147	147	147	147	147
AdvanceSchedule	SW	1	266	206	206	206	206	206
	NS		321	286	286	286	286	286
	KL	2	225	163	163	163	163	163
StartSchedule			161	161	161	161	161	161
StopSchedule			125	125	125	125	125	125
GetScheduleStatus			153	153	153	153	153	153
GetScheduleValue			123	123	123	123	123	123
GetScheduleNext			58	58	58	58	58	58
SetScheduleNext			110	110	110	110	110	110
GetArrivalpointDelay			83	83	83	83	83	83
SetArrivalpointDelay			94	94	94	94	94	94
GetArrivalpointTasksetRef			79	79	79	79	79	79
GetArrivalpointNext			83	83	83	83	83	83
SetArrivalpointNext			122	122	122	122	122	122
TestArrivalpointWritable			95	95	95	95	95	95
GetExecutionTime			79	79	79	79	79	79
GetLargestExecutionTime			78	78	78	78	78	78

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
ResetLargestExecutionTime			67	67	67	67	67	67
GetStackOffset			18	18	18	18	18	18

Notes

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

Number	Note
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	110	154	215	113	155	228
	NS	100	147	196	103	148	209
	KL	45	91	147	48	95	174
TerminateTask	LExt	0	0	0	0	0	0
	H	154	152	159	152	152	159
ChainTask	SWL	245	286	387	287	330	428
	SWH	306	340	440	347	383	482
	NSL	245	286	387	287	330	428
	NSH	301	335	435	342	378	477
Schedule	SW	97	95	110	95	95	110

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetTaskID		35	34	34	34	34	34
GetTaskState		110	109	110	120	121	120
EnableAllInterrupts		29	29	29	29	29	29
DisableAllInterrupts		47	47	47	47	47	47
ResumeAllInterrupts		37	37	37	37	37	37
SuspendAllInterrupts		55	55	55	55	55	55
ResumeOSInterrupts		37	37	37	37	37	37
SuspendOSInterrupts		55	55	55	55	55	55
GetResource	Task	45	45	46	42	42	43
	Combined	87	87	87	84	84	84
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	87	87	87	85	85	85
	Combined	103	103	103	100	100	100
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	134	134	129
	NS	n/a	n/a	n/a	134	134	126
	KL	n/a	n/a	n/a	77	77	81
ClearEvent		n/a	n/a	n/a	56	56	56
GetEvent		n/a	n/a	n/a	38	38	38
WaitEvent	<default>	n/a	n/a	n/a	359	370	434
	fp	n/a	n/a	n/a	368	380	443
GetAlarmBase		99	103	103	99	99	99
GetAlarm		118	121	121	118	118	118
SetRelAlarm		130	133	133	130	130	130
SetAbsAlarm		121	124	124	121	121	121
CancelAlarm		93	94	94	93	93	93
InitCounter		111	113	113	111	111	111
GetCounterValue		97	99	99	97	97	97
osek_tick_alarm	<default>	123	124	124	123	123	123
	KL	54	55	55	54	54	54
osek_incr_counter		13	14	14	13	13	13
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		332	331	331	331	331	331
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	56	56	56	56	56	56
InitCOM		18	18	18	21	21	21

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
CloseCOM		18	18	18	18	18	18
StartCOM		54	54	54	249	249	249
StopCOM		26	26	26	26	26	26
ReadFlag		n/a	n/a	n/a	14	14	14
ResetFlag		n/a	n/a	n/a	10	10	10
ReceiveMessage		90	92	92	242	242	242
GetMessageResource		n/a	n/a	n/a	95	95	95
ReleaseMessageResource		n/a	n/a	n/a	132	132	132
GetMessageStatus		n/a	n/a	n/a	52	52	52
SendMessage	SW	238	284	345	407	449	522
	NS	225	274	323	394	439	500
	KL	116	164	220	291	338	417
ActivateTaskset	SW	86	629	706	92	634	730
	NS	78	616	693	84	621	717
	KL	28	570	647	34	577	671
	SW2	86	629	706	92	634	730
	NS2	78	616	693	84	621	717
	KL2	28	570	647	34	577	671
ChainTaskset	SWL	234	743	863	272	788	918
	SWH	300	670	787	337	717	826
	NSL	234	743	863	272	788	918
	NSH	295	665	782	332	712	821
GetTasksetRef		35	34	35	34	35	34
MergeTaskset		90	90	90	90	90	90
AssignTaskset		33	33	33	33	33	33
RemoveTaskset		89	89	89	89	89	89
TestSubTaskset		103	103	103	103	103	103
TestEquivalentTaskset		96	96	96	96	96	96
TickSchedule	SW	169	736	806	198	755	852
	NS	154	721	791	183	740	837
	KL	98	665	735	127	684	781
	SW2	169	735	812	198	742	836
	NS2	154	720	797	183	727	821
	KL2	98	664	741	127	671	765
AdvanceSchedule	SW	154	714	784	176	733	830
	NS	139	699	769	161	718	815

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	KL	93	655	725	117	674	771
	SW2	154	713	790	176	720	814
	NS2	139	698	775	161	705	799
	KL2	93	654	731	117	661	755
StartSchedule		119	119	119	119	119	119
StopSchedule		101	101	101	101	101	101
GetScheduleStatus		111	111	111	111	111	111
GetScheduleValue		108	108	108	108	108	108
GetScheduleNext		37	37	37	37	37	37
SetScheduleNext		34	34	34	34	34	34
GetArrivalpointDelay		34	34	34	34	34	34
SetArrivalpointDelay		30	30	30	30	30	30
GetArrivalpointTasksetRef		30	30	30	30	30	30
GetArrivalpointNext		33	33	33	33	33	33
SetArrivalpointNext		30	30	30	30	30	30
TestArrivalpointWritable		39	39	39	39	39	39
GetExecutionTime		16	16	16	16	16	16
GetLargestExecutionTime		31	30	30	30	30	30
ResetLargestExecutionTime		19	18	18	18	18	18
GetStackOffset		34	34	34	34	34	34

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	111	156	217	114	158	232
	NS	101	149	198	104	151	213
	KL	46	93	149	49	98	176
TerminateTask	LExt	0	0	0	0	0	0
	H	339	334	341	334	334	341
ChainTask	SWL	465	504	619	506	553	663
	SWH	520	552	666	560	600	711

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NSL	465	504	619	506	553	663
	NSH	515	547	661	555	595	706
Schedule	SW	99	97	113	97	97	113
GetTaskID		35	34	34	34	34	34
GetTaskState		111	110	111	122	123	122
EnableAllInterrupts		29	29	29	29	29	29
DisableAllInterrupts		47	47	47	47	47	47
ResumeAllInterrupts		37	37	37	37	37	37
SuspendAllInterrupts		55	55	55	55	55	55
ResumeOSInterrupts		37	37	37	37	37	37
SuspendOSInterrupts		55	55	55	55	55	55
GetResource	Task	45	45	46	42	42	43
	Combined	87	87	87	84	84	84
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	87	87	87	85	85	85
	Combined	103	103	103	100	100	100
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	134	134	129
	NS	n/a	n/a	n/a	134	134	126
	KL	n/a	n/a	n/a	77	77	81
ClearEvent		n/a	n/a	n/a	56	56	56
GetEvent		n/a	n/a	n/a	38	38	38
WaitEvent	<default>	n/a	n/a	n/a	527	541	603
	fp	n/a	n/a	n/a	536	551	612
GetAlarmBase		99	103	103	99	99	99
GetAlarm		118	121	121	118	118	118
SetRelAlarm		130	133	133	130	130	130
SetAbsAlarm		121	124	124	121	121	121
CancelAlarm		93	94	94	93	93	93
InitCounter		111	113	113	111	111	111
GetCounterValue		97	99	99	97	97	97
osek_tick_alarm	<default>	123	124	124	123	123	123
	KL	54	55	55	54	54	54
osek_incr_counter		13	14	14	13	13	13
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		631	630	630	629	629	629

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	56	56	56	56	56	56
InitCOM		18	18	18	21	21	21
CloseCOM		18	18	18	18	18	18
StartCOM		54	54	54	249	249	249
StopCOM		26	26	26	26	26	26
ReadFlag		n/a	n/a	n/a	14	14	14
ResetFlag		n/a	n/a	n/a	10	10	10
ReceiveMessage		90	92	92	242	242	242
GetMessageResource		n/a	n/a	n/a	96	96	96
ReleaseMessageResource		n/a	n/a	n/a	132	132	132
GetMessageStatus		n/a	n/a	n/a	52	52	52
SendMessage	SW	239	286	347	408	452	526
	NS	226	276	325	395	442	504
	KL	117	166	222	292	341	419
ActivateTaskset	SW	86	629	706	92	634	730
	NS	78	616	693	84	621	717
	KL	28	570	647	34	577	671
	SW2	86	629	706	92	634	730
	NS2	78	616	693	84	621	717
	KL2	28	570	647	34	577	671
ChainTaskset	SWL	454	958	1092	491	1008	1149
	SWH	513	879	1010	549	930	1051
	NSL	454	958	1092	491	1008	1149
	NSH	508	874	1005	544	925	1046
GetTasksetRef		35	34	35	34	35	34
MergeTaskset		90	90	90	90	90	90
AssignTaskset		33	33	33	33	33	33
RemoveTaskset		89	89	89	89	89	89
TestSubTaskset		103	103	103	103	103	103
TestEquivalentTaskset		96	96	96	96	96	96
TickSchedule	SW	169	736	806	198	755	852
	NS	154	721	791	183	740	837
	KL	98	665	735	127	684	781
	SW2	169	735	812	198	742	836
	NS2	154	720	797	183	727	821

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	KL2	98	664	741	127	671	765
AdvanceSchedule	SW	154	714	784	176	733	830
	NS	139	699	769	161	718	815
	KL	93	655	725	117	674	771
	SW2	154	713	790	176	720	814
	NS2	139	698	775	161	705	799
	KL2	93	654	731	117	661	755
StartSchedule		119	119	119	119	119	119
StopSchedule		101	101	101	101	101	101
GetScheduleStatus		111	111	111	111	111	111
GetScheduleValue		108	108	108	108	108	108
GetScheduleNext		37	37	37	37	37	37
SetScheduleNext		34	34	34	34	34	34
GetArrivalpointDelay		34	34	34	34	34	34
SetArrivalpointDelay		30	30	30	30	30	30
GetArrivalpointTasksetRef		30	30	30	30	30	30
GetArrivalpointNext		33	33	33	33	33	33
SetArrivalpointNext		30	30	30	30	30	30
TestArrivalpointWritable		39	39	39	39	39	39
GetExecutionTime		111	111	111	110	110	110
GetLargestExecutionTime		44	42	42	42	42	42
ResetLargestExecutionTime		31	29	29	29	29	29
GetStackOffset		34	34	34	34	34	34

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	516	537	598	498	540	612
	NS	546	566	623	526	569	635
	KL	443	465	521	425	468	535
TerminateTask	LExt	386	380	387	380	380	387

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
	H	429	422	429	422	422	429
ChainTask	SWL	943	952	1067	961	1006	1129
	SWH	993	1002	1116	1010	1050	1157
	NSL	988	1000	1116	1006	1055	1177
	NSH	1032	1041	1156	1049	1090	1200
Schedule	SW	149	146	159	146	146	159
GetTaskID		43	42	42	42	42	42
GetTaskState		525	504	505	512	513	512
EnableAllInterrupts		37	37	37	37	37	37
DisableAllInterrupts		55	55	55	55	55	55
ResumeAllInterrupts		51	51	51	51	51	51
SuspendAllInterrupts		63	63	63	63	63	63
ResumeOSInterrupts		51	51	51	51	51	51
SuspendOSInterrupts		63	63	63	63	63	63
GetResource	Task	889	863	433	924	927	505
	Combined	409	409	409	484	484	484
	CLEx	449	445	445	522	522	522
ReleaseResource	Task	413	413	413	488	488	488
	Combined	387	387	387	463	463	463
	CLEx	416	413	413	491	491	491
SetEvent	SW	n/a	n/a	n/a	517	517	517
	NS	n/a	n/a	n/a	546	546	546
	KL	n/a	n/a	n/a	476	476	464
ClearEvent		n/a	n/a	n/a	86	86	86
GetEvent		n/a	n/a	n/a	433	433	433
WaitEvent	<default>	n/a	n/a	n/a	605	620	677
	fp	n/a	n/a	n/a	614	630	686
GetAlarmBase		385	402	402	385	385	385
GetAlarm		391	407	407	391	391	391
SetRelAlarm		426	446	446	426	426	426
SetAbsAlarm		419	438	438	419	419	419
CancelAlarm		377	391	391	377	377	377
InitCounter		653	687	687	653	653	653
GetCounterValue		343	356	356	343	343	343
osek_tick_alarm	<default>	157	158	158	157	157	157
	KL	54	55	55	54	54	54

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
osek_incr_counter		13	14	14	13	13	13
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		657	656	656	655	655	655
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	60	60	60	60	60	60
InitCOM		18	18	18	21	21	21
CloseCOM		18	18	18	18	18	18
StartCOM		66	66	66	261	261	261
StopCOM		37	37	37	37	37	37
ReadFlag		n/a	n/a	n/a	37	37	37
ResetFlag		n/a	n/a	n/a	34	34	34
ReceiveMessage		287	296	296	433	433	433
GetMessageResource		n/a	n/a	n/a	731	731	731
ReleaseMessageResource		n/a	n/a	n/a	677	677	677
GetMessageStatus		n/a	n/a	n/a	214	214	214
SendMessage	SW	837	867	928	972	1014	1086
	NS	864	893	950	997	1040	1106
	KL	707	738	794	853	896	963
ActivateTaskset	SW	499	899	971	506	896	985
	NS	577	976	1032	583	973	1046
	KL	438	838	908	445	835	922
	SW2	499	899	971	506	896	985
	NS2	577	976	1032	583	973	1046
	KL2	438	838	908	445	835	922
ChainTaskset	SWL	967	1369	1498	1009	1411	1544
	SWH	1110	1504	1635	1151	1551	1677
	NSL	1056	1457	1572	1097	1499	1618
	NSH	1146	1537	1668	1186	1584	1710
GetTasksetRef		419	398	399	398	399	398
MergeTaskset		163	163	163	163	163	163
AssignTaskset		85	85	85	85	85	85
RemoveTaskset		159	159	159	159	159	159
TestSubTaskset		173	173	173	173	173	173
TestEquivalentTaskset		164	164	164	164	164	164
TickSchedule	SW	272	1062	1123	666	1088	1173
	NS	347	1099	1160	703	1125	1210

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
AdvanceSchedule	KL	192	990	1051	594	1016	1101
	SW2	272	1060	1130	666	1057	1144
	NS2	347	1097	1167	703	1094	1181
	KL2	192	988	1058	594	985	1072
	SW	257	1050	1111	654	1076	1161
	NS	313	1108	1169	712	1134	1219
	KL	189	977	1038	581	1003	1088
	SW2	257	1048	1118	654	1045	1132
	NS2	313	1106	1176	712	1103	1190
	KL2	189	975	1045	581	972	1059
StartSchedule		181	181	181	181	181	181
StopSchedule		149	149	149	149	149	149
GetScheduleStatus		169	169	169	169	169	169
GetScheduleValue		166	166	166	166	166	166
GetScheduleNext		59	59	59	59	59	59
SetScheduleNext		90	90	90	90	90	90
GetArrivalpointDelay		64	64	64	64	64	64
SetArrivalpointDelay		67	67	67	67	67	67
GetArrivalpointTasksetRef		53	53	53	53	53	53
GetArrivalpointNext		57	57	57	57	57	57
SetArrivalpointNext		88	88	88	88	88	88
TestArrivalpointWritable		63	63	63	63	63	63
GetExecutionTime		143	143	143	142	142	142
GetLargestExecutionTime		416	395	395	395	395	395
ResetLargestExecutionTime		395	374	374	374	374	374
GetStackOffset		34	34	34	34	34	34

4.3.2 OS Start-up Time

OS start-up time is the time from the entry to the `StartOS()` function to the execution of the first instruction in a user task (including the idle task) without any hook routines being called. This time is always application dependent, since `StartOS()` may activate any number of tasks and start any number of user-specified alarms.

4.3.3 Interrupt Latencies

Interrupt latency is the time between an interrupt request being recognized by the target hardware and the execution of the first instruction of the user provided handler function. The following tables give the interrupt latencies (in CPU cycles).

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	27	27	27	27	27	27
	Cat 2	36	36	36	36	36	36

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	27	27	27	27	27	27
	Cat 2	173	173	173	172	172	172

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	27	27	27	27	27	27
	Cat 2	173	173	173	172	172	172

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.

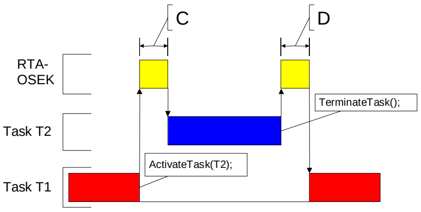

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

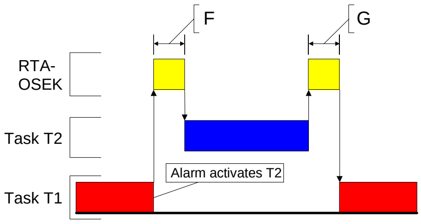

Figure 4: Task Activation from an Alarm

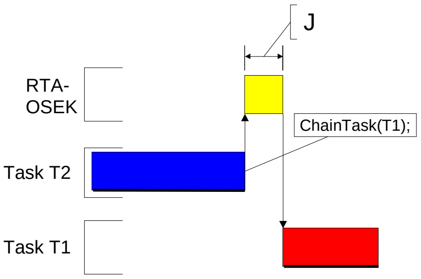

Figure 2: Task Chaining

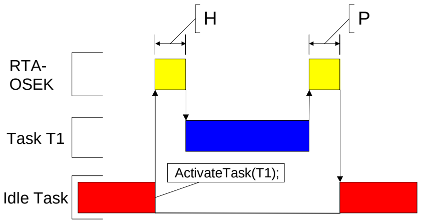

Figure 3: Task Activation from Idle Task

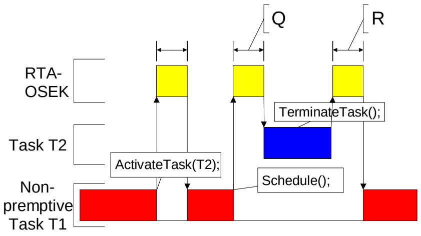

Figure 5: Non-Premptive Task Calls Schedule()

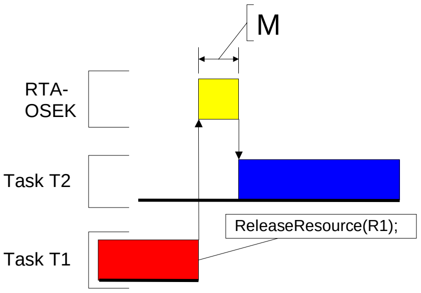

Figure 6: Blocked Task Activated by ReleaseResource()

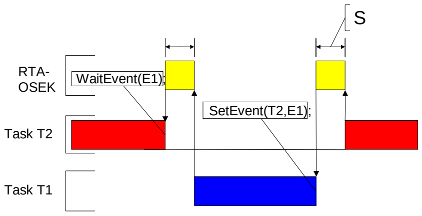

Figure 7: Waiting Task Activated by SetEvent()

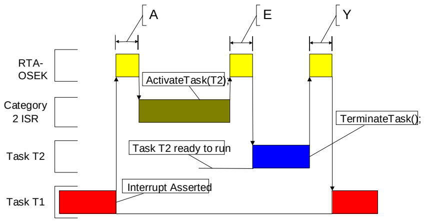

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	89	127	165	88	127	170
Figure 1: D	Heavy, Basic/Extended	154	182	226	192	196	230
ChainTask	Light, Basic	190	250	344	188	244	349
Figure 2: J	Heavy, Basic/Extended	426	506	644	461	515	654
Pre-emption	Light, Basic	171	232	326	171	232	364
Figure 1: C	Heavy, Basic/Extended	230	271	384	273	320	434
From idle task	Light, Basic	171	232	326	171	232	364
Figure 3: H	Heavy, Basic/Extended	230	271	384	273	320	434
Triggered by alarm	Light, Basic	304	364	458	303	364	496
Figure 4: F	Heavy, Basic/Extended	363	403	516	405	452	566
Schedule	Light, Basic	150	161	222	146	162	226
Figure 5: Q	Heavy, Basic/Extended	209	205	270	248	254	308
Release resource	Light, Basic	160	172	220	154	169	220
Figure 6: M	Heavy, Basic/Extended	219	216	268	256	261	302
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	419	432	602
From category 2 ISR	Light, Basic	129	141	189	126	141	192
Figure 8: E	Heavy, Basic/Extended	188	185	237	228	233	274

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	281	304	339	277	304	347
Figure 1: D	Heavy, Basic/Extended	338	348	395	359	366	397
ChainTask	Light, Basic	416	469	576	411	462	580
Figure 2: J	Heavy, Basic/Extended	830	885	1039	845	897	1046
Pre-emption	Light, Basic	288	343	449	286	341	486
Figure 1: C	Heavy, Basic/Extended	342	383	510	385	435	562
From idle task	Light, Basic	290	345	451	288	343	488
Figure 3: H	Heavy, Basic/Extended	344	385	512	387	437	564
Triggered by alarm	Light, Basic	421	475	581	418	473	618
Figure 4: F	Heavy, Basic/Extended	475	515	642	517	567	694
Schedule	Light, Basic	269	272	345	263	275	352
Figure 5: Q	Heavy, Basic/Extended	322	315	394	361	370	436
Release resource	Light, Basic	277	283	343	269	278	342
Figure 6: M	Heavy, Basic/Extended	330	326	392	367	373	426
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	514	527	710
From category 2 ISR	Light, Basic	431	434	494	423	432	496
Figure 8: E	Heavy, Basic/Extended	484	477	543	521	527	580

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	387	409	443	381	409	452
Figure 1: D	Heavy, Basic/Extended	428	436	484	447	455	485
ChainTask	Light, Basic	894	917	1024	866	910	1034
Figure 2: J	Heavy, Basic/Extended	1393	1423	1578	1383	1436	1585
Pre-emption	Light, Basic	688	720	826	665	718	861
Figure 1: C	Heavy, Basic/Extended	743	760	887	765	813	938
From idle task	Light, Basic	690	722	828	667	720	863
Figure 3: H	Heavy, Basic/Extended	745	762	889	767	815	940
Triggered by alarm	Light, Basic	855	886	992	831	884	1027
Figure 4: F	Heavy, Basic/Extended	910	926	1053	931	979	1104
Schedule	Light, Basic	313	315	387	306	319	395
Figure 5: Q	Heavy, Basic/Extended	366	358	436	404	414	479
Release resource	Light, Basic	573	579	640	643	652	717
Figure 6: M	Heavy, Basic/Extended	626	622	689	741	747	801
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	897	910	1095
From category 2 ISR	Light, Basic	463	466	526	455	464	528
Figure 8: E	Heavy, Basic/Extended	516	509	575	553	559	612

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	24	24	27	24	24	27
BCC1 lightweight, floating-point	27	27	30	27	27	30
BCC1 heavyweight, integer	32	32	35	32	32	35
BCC1 heavyweight, floating-point	32	32	35	32	32	35
BCC2 lightweight, integer	n/a	27	32	n/a	27	32
BCC2 lightweight, floating-point	n/a	27	32	n/a	27	32
BCC2 heavyweight, integer	n/a	32	39	n/a	32	39
BCC2 heavyweight, floating-point	n/a	32	39	n/a	32	39
ECC1 heavyweight, integer	n/a	n/a	n/a	42	42	45
ECC1 heavyweight, floating-point	n/a	n/a	n/a	42	42	45
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	49
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	49
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	27	27	27	27	27	27
BCC1 lightweight, floating-point	30	30	30	30	30	30
BCC1 heavyweight, integer	35	35	35	35	35	35
BCC1 heavyweight, floating-point	35	35	35	35	35	35
BCC2 lightweight, integer	n/a	30	32	n/a	30	32
BCC2 lightweight, floating-point	n/a	30	32	n/a	30	32
BCC2 heavyweight, integer	n/a	35	39	n/a	35	39
BCC2 heavyweight, floating-point	n/a	35	39	n/a	35	39
ECC1 heavyweight, integer	n/a	n/a	n/a	45	45	45
ECC1 heavyweight, floating-point	n/a	n/a	n/a	45	45	45
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	49
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	49

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		33	33	36	33	33	36
BCC1 lightweight, floating-point		36	36	39	36	36	39
BCC1 heavyweight, integer		41	41	44	41	41	44
BCC1 heavyweight, floating-point		41	41	44	41	41	44
BCC2 lightweight, integer		n/a	36	41	n/a	36	41
BCC2 lightweight, floating-point		n/a	36	41	n/a	36	41
BCC2 heavyweight, integer		n/a	41	48	n/a	41	48
BCC2 heavyweight, floating-point		n/a	41	48	n/a	41	48
ECC1 heavyweight, integer		n/a	n/a	n/a	51	51	54
ECC1 heavyweight, floating-point		n/a	n/a	n/a	51	51	54
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
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		36	36	36	36	36	36
BCC1 lightweight, floating-point		39	39	39	39	39	39
BCC1 heavyweight, integer		44	44	44	44	44	44
BCC1 heavyweight, floating-point		44	44	44	44	44	44
BCC2 lightweight, integer		n/a	39	41	n/a	39	41
BCC2 lightweight, floating-point		n/a	39	41	n/a	39	41
BCC2 heavyweight, integer		n/a	44	48	n/a	44	48
BCC2 heavyweight, floating-point		n/a	44	48	n/a	44	48
ECC1 heavyweight, integer		n/a	n/a	n/a	54	54	54
ECC1 heavyweight, floating-point		n/a	n/a	n/a	54	54	54
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

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	33	33	36	33	33	36
BCC1 lightweight, floating-point	36	36	39	36	36	39
BCC1 heavyweight, integer	41	41	44	41	41	44
BCC1 heavyweight, floating-point	41	41	44	41	41	44
BCC2 lightweight, integer	n/a	36	41	n/a	36	41
BCC2 lightweight, floating-point	n/a	36	41	n/a	36	41
BCC2 heavyweight, integer	n/a	41	48	n/a	41	48
BCC2 heavyweight, floating-point	n/a	41	48	n/a	41	48
ECC1 heavyweight, integer	n/a	n/a	n/a	51	51	54
ECC1 heavyweight, floating-point	n/a	n/a	n/a	51	51	54
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
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	36	36	36	36	36	36
BCC1 lightweight, floating-point	39	39	39	39	39	39
BCC1 heavyweight, integer	44	44	44	44	44	44
BCC1 heavyweight, floating-point	44	44	44	44	44	44
BCC2 lightweight, integer	n/a	39	41	n/a	39	41
BCC2 lightweight, floating-point	n/a	39	41	n/a	39	41
BCC2 heavyweight, integer	n/a	44	48	n/a	44	48
BCC2 heavyweight, floating-point	n/a	44	48	n/a	44	48
ECC1 heavyweight, integer	n/a	n/a	n/a	54	54	54
ECC1 heavyweight, floating-point	n/a	n/a	n/a	54	54	54
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

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.