RTA-OSEK

Binding Manual: Diab56x

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: M00086-001

Disclaimer

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

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

Trademarks

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

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

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

Contents

1 About this Guide

This guide provides target-specific information for the Diab56x port of LiveDevices' RTA-OSEK. It supplements the more general information in the *RTA-OSEK User Guide*.

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

1.1 Who Should Read this Guide?

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

1.2 Conventions

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

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

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

2 Toolchain Issues

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

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

The Diab56x supports the single flat memory model supported by the Wind River Systems, Inc. (Diab) toolchain. This toolchain supports the Embedded Application Binary Interface, EABI.

Target hardware specific options in the toolchain may be selected by setting the DOS environment variable `CPU_TYPE` which is passed as an argument to the toolchain using the `-t` option.

2.1 Compiler

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

Vendor	Wind River Systems, Inc.
Compiler	Wind River (Diab) Compiler for PowerPC
Version	5.5.1.0

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

Option	Description
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.

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`	Stop after the assembly step and produce an object file with default file extension .o.
`-g0`	Turn debug mode off.
`-Xsmall-data=0`	Place small non-constant static and global variables with a size in bytes less than or equal to n in the SDATA section class.
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.

2.2 Assembler

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

Vendor	Wind River Systems, Inc.
Assembler	Wind River (Diab) Assembler for the PowerPC
Version	5.5.1.0

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

Option	Description
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.

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

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

Option	Description
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.

2.3 Linker/Locator

The compulsory linker/locator options for an RTA-OSEK application are shown in the following table:

Option	Description
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.

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

Sections	ROM/RAM	Description
`os_pid`	ROM	RTA-OSEK read-only data. For performance reasons, it can be mapped into RAM (if initialized correctly).
`os_pidf`	ROM	RTA-OSEK read-only data. This section contains RTA-OSEK constant data, all of which is far-addressed (OS_CONST_VAR and OS_CONST_ROM). For performance reasons, it can be mapped into 'far' RAM (if initialized correctly).
`os_pird`	ROM	RTA-OSEK initialization data. This is only accessed during StartOS(). It will normally be located in slow ROM.
`os_pnird`	ROM	RTA-OSEK near initialization data. This is only accessed during StartOS(). It will normally be located in slow ROM.
`os_text`	ROM	RTA-OSEK code section.
`os_intvec`	ROM	Vector table if generated by RTA-OSEK
`os_vec_etr`	ROM	RTA-OSEK RCPU vector table for use when the BBCMCR[ETRE] bit is 1
`os_pir`	RAM	RTA-OSEK initialized data. Must be initialized during C-startup.

Sections	ROM/RAM	Description
os_pnir	RAM	RTA-OSEK near initialized data. Must be initialized during C-startup. Must be placed in the compiler SDA (near addressing)
os_pur	RAM	RTA-OSEK uninitialized data. Must be zeroed during C-startup.
os_trace_ram	RAM	RTA-TRACE buffer. RTA-TRACE buffer. Can be located in 'far' RAM. Does not need to be initialized.

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

C Library Functions	Description
crt0.s	startup code with automatic initialization

2.4 Debugger

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

ORTI compatible debuggers	Lauterbach TRACE32

The RTA-OSEK GUI outputs a file with the extension `.ort`. This file should be loaded into the debugger with the command `Task.ORTI <file>`. Please refer to the debugger documentation for further details on its support for ORTI.

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for Diab56x. 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 *MPC56x Reference Manual*.

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

IPL Value	SIMASK2 / SIMASK3	Description
0	SIMASK3 bits 16–31 = 0	User level
1	SIMASK3 bits 15–31 = 0	IMB_IRQ31
2	SIMASK3 bits 14–31 = 0	IMB_IRQ30
3	SIMASK3 bits 13–31 = 0	IMB_IRQ29
4	SIMASK3 bits 12–31 = 0	IMB_IRQ28
5	SIMASK3 bits 11–31 = 0	Level 7
6	SIMASK3 bits 10–31 = 0	EXT_IRQ 7
…	…	…
16	SIMASK3 bits 0–31 = 0	IMB_IRQ20
17	SIMASK2 bit 31 = 0, SIMASK3 bits 0–31 = 0	Level 5
…	…	…
45	SIMASK2 bits 3–31 = 0, SIMASK3 bits 0–31 = 0	IMB_IRQ1
46	SIMASK2 bits 2–31 = 0, SIMASK3 bits 0–31 = 0	IMB_IRQ0
47	SIMASK2 bits 1–31 = 0, SIMASK3 bits 0–31 = 0	Level 0 Interrupt
48	MSR[EE] bit clear	Category 1 Interrupts
49	Not applicable	Synchronous Exceptions

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 Table	Constraints
RCPU	Category 1 ISRs only
EEIR	Category 2 ISRs and Category 1 ISRs

The valid base addresses for the vector table are:

Base Address	Notes
set by EIBADR	EEIR vector table
0x00000100	RCPU vector table (MSR[IP] set to 0, BBCMCR[ETRE] set to 0)
0xFFF00100	RCPU vector table (MSR[IP] set to 1, BBCMCR[ETRE] set to 0)

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 Wind River Systems, Inc. C compiler can generate appropriate interrupt handling code for a C function decorated with the __interrupt__ function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

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

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

The following table shows the syntax for labels attached to RTA-OSEK Category 2 interrupt handlers (VVVVV represents the 5 hex digit, upper-case, zero-padded value of the vector location).

Vector File	Notes
osveclow.s	Contains vectors located from 0x0000200 upwards. MSR[IP] = 0 and BBCMCR[ETRE] = 0
osvechi.s	Contains vectors located from 0xFFF0200 upwards. MSR[IP] = 1 and BBCMCR[ETRE] = 0
osvecetr.s	Contains a relocatable vector segment. This should be located by the linker command file. BBCMCR[ETRE] = 1

3.1.6 Category 2 Interrupt Pre-emption

Category 2 interrupts are run with the recoverable exception bit of the machine status register set to 1 (MSR[RI]) to allow debugging exceptions and higher priority interrupts to safely pre-empt them.

3.1.7 Floating-point

Tasks and Category 2 ISRs inherit the MSR[FP] bit setting from the environment they preempt. Thus, if MSR[FP] is initialized to 1 before calling StartOS, all tasks and Category 2 ISRs can use the PowerPC floating-point hardware.

In contrast, any Category 1 interrupt handler that uses the floating-point hardware must re-enable MSR[FP] before using the floating-point hardware. This is because the PowerPC exception handling mechanism disables floating-point.

3.2 Register Settings

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

Register	Required Value	Notes
SIUMCR[EIC] bit	1	
SIUMCR[LPMASK_EN] bit	0	
BBCMCR[EIR] bit	1	See section 3.1.5
MSR[IP]	1	See section 3.1.5
MSR[FP]	0/1	Must be 1 if hardware floating-point is used
EIBADR	Address of EEIR vector table	

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

Register	Notes
SIMASK2	Interrupt mask register.
SIMASK3	Interrupt mask register.
MSR[EE] bit	Global interrupt enable bit.

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 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): 64

Timing

API max usage (bytes): 80

Extended

API max usage (bytes): 96

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

3.3.3 Stack discipline

RTA-OSEK adheres to the EABI requirements for stack discipline, in particular that the stack pointer (R1) is adjusted only once in each routine, a back-link is maintained, and the stack pointer is kept aligned to a 16-byte boundary.

During start-up the user's application code should set R1 to a suitable value, in on-chip or external RAM.

Important: The initial stack pointer (R1) value must be made known in the symbol os_SP_INIT.

Typically your linker control file will need to include the line:
os_SP_INIT = __SP_INIT;

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	**MPC565**
Clock speed (MHz)	40
Code memory	On-chip FLASH
Read-only data memory	On-chip FLASH
Read-write data memory	On-chip RAM

4.1 Functionality

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

Configuration	**Application Uses**					
Events	**No**			**Yes**		
Shared Task Priorities	**No**		**Yes**	**No**		**Yes**
Multiple Task Activations	**No**	**Yes**		**No**	**Yes**	
Maximum number of tasks	32	32	32	32	32	32
Maximum number of not suspended tasks	32	32	32	32	32	32
Maximum number of priorities	32	32	32	32	32	32
Number of tasks per priority (for BCC2 and ECC2)	n/a	32	32	n/a	32	32
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	32	32	32
Limits for the number of alarm objects (per system / per task)	not limited by RTA-OSEK					
Limits for the number of standard resources (per system)	255	255	255	255	255	255
Limits for the number of internal resources (per system)	not limited by RTA-OSEK					
Limits for the number of nested resources (per system / per task)	255	255	255	255	255	255
Limits for the number of application modes	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	40	40	40	40	40	40
	ROM	152	152	156	280	280	284
COM overhead	RAM	2	2	2	2	2	2
	ROM	9	9	9	9	9	9

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	60	60	60	60	60	60
	ROM	224	224	228	352	352	356
COM overhead	RAM	2	2	2	2	2	2
	ROM	9	9	9	9	9	9

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	82	82	82	82	82	82
	ROM	276	276	280	404	404	408
COM overhead	RAM	2	2	2	2	2	2
	ROM	9	9	9	9	9	9

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	40	40	40	40	40	40
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	44	44	44	44	44	44
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	52	60	n/a	52	60
ECC1, Integer task	RAM	n/a	n/a	n/a	116	116	116
	ROM	n/a	n/a	n/a	64	64	64
ECC1, floating-point task	RAM	n/a	n/a	n/a	236	236	236
	ROM	n/a	n/a	n/a	64	64	64
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	118
	ROM	n/a	n/a	n/a	n/a	n/a	72
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	238
	ROM	n/a	n/a	n/a	n/a	n/a	72
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	40	40	40	40	40	40

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Category 2 ISR, floating-point	RAM	120	120	120	120	120	120
	ROM	76	76	76	76	76	76
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
Counter	RAM	4	4	4	4	4	4
	ROM	148	148	148	148	148	148
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	2	2	2	2	2	2
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	124	124	124	124	124	124
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	52	52	52	52	52	52
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	56	56	56	56	56	56
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	64	72	n/a	64	72
ECC1, Integer task	RAM	n/a	n/a	n/a	128	128	128
	ROM	n/a	n/a	n/a	76	76	76
ECC1, floating-point task	RAM	n/a	n/a	n/a	248	248	248
	ROM	n/a	n/a	n/a	76	76	76
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	130
	ROM	n/a	n/a	n/a	n/a	n/a	84
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	250
	ROM	n/a	n/a	n/a	n/a	n/a	84
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	116	116	116	116	116	116
Category 2 ISR, floating-point	RAM	132	132	132	132	132	132
	ROM	148	148	148	148	148	148
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	48	48	48	48	48	48
Counter	RAM	4	4	4	4	4	4
	ROM	148	148	148	148	148	148
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	2	2	2	2	2	2
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	124	124	124	124	124	124
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
	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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	14	14	14	14	14	14
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	68	76	n/a	68	76
ECC1, Integer task	RAM	n/a	n/a	n/a	132	132	132
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	252	252	252
	ROM	n/a	n/a	n/a	80	80	80
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	134
	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	254
	ROM	n/a	n/a	n/a	n/a	n/a	88
Category 2 ISR	RAM	14	14	14	14	14	14
	ROM	124	124	124	124	124	124
Category 2 ISR, floating-point	RAM	134	134	134	134	134	134
	ROM	156	156	156	156	156	156
Resource	RAM	16	16	16	16	16	16
	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	16	16	16	16	16	16
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	52	52	52	52	52	52
Counter	RAM	4	4	4	4	4	4

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
	ROM	152	152	152	152	152	152
Message	RAM	11	11	11	31	31	31
	ROM	24	24	24	60	60	60
Flag	RAM	2	2	2	2	2	2
	ROM	1	1	1	1	1	1
Message resource	RAM	16	16	16	16	16	16
	ROM	28	28	28	28	28	28
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	124	124	124	124	124	124
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Arrivalpoint (writable)	RAM	20	20	20	20	20	20
	ROM	20	20	20	20	20	20
Schedule	RAM	20	20	20	20	20	20
	ROM	44	44	44	44	44	44
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

4.2.3 Size of Linkable Modules

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

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

The call variants are as follows:

Variant	Description
1i	Idle task is only ECC task.
CCCA	OSEK COM class.
CCCB	OSEK COM class.
CLEx	Resource tests in Extended OS Status.
fp	ECC task uses floating-point.
H	Used for heavyweight termination only.
Hook	Pre- and Post- Task hooks are used.
KL	API is called from OS level.
KL1i	API is called from OS level, idle task is only ECC task.
KL2	Activated taskset has one BCC2 task.
LExt	Used for lightweight termination in Extended Status.
ServiceID	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
Parameters	`ErrorHook` uses `GetServiceID` and `GetServiceParameters`.
NoHook	Pre- and/or Post- Task hooks are not used.
NS	No context switch is possible.
NS1i	No context switch is possible, idle task is only ECC task.
NS2	Activated taskset has one BCC2 task.
NSH	Chain from heavyweight task, not to higher priority.
NSL	Chain from lightweight task, not to higher priority.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.
Task	Resource is used only by tasks.

Standard

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	308	400	448	316	408	488
	NS		252	352	400	260	360	440
	KL	2	116	212	264	124	220	300
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	20	20	20	20	20	20
ChainTask	SWL	1, 8	236	332	380	244	340	420
	SWH	1, 9	280	372	420	288	380	460
	NSL	8	236	332	380	244	340	420
	NSH	9	272	364	412	280	372	452
Schedule			188	188	228	188	188	228
GetTaskID			44	44	44	44	44	44
GetTaskState			172	172	172	196	196	196
EnableAllInterrupts			32	32	32	32	32	32
DisableAllInterrupts			36	36	36	36	36	36
ResumeAllInterrupts			60	60	60	60	60	60
SuspendAllInterrupts			84	84	84	84	84	84
ResumeOSInterrupts			96	96	96	96	96	96
SuspendOSInterrupts			128	128	128	128	128	128
GetResource	Task	7	32	32	40	32	32	40
	Combined	6	184	184	184	184	184	184
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	200	200	200	200	200	200
	Combined	6	396	396	396	396	396	396
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	256	256	360
	NS		n/a	n/a	n/a	176	176	312
	NS1i	10	n/a	n/a	n/a	128	n/a	n/a
	KL	2	n/a	n/a	n/a	76	76	188
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	100	100	100
GetEvent			n/a	n/a	n/a	20	20	20
WaitEvent	<default>		n/a	n/a	n/a	400	400	704
	fp	11	n/a	n/a	n/a	444	444	812
	1i	10	n/a	n/a	n/a	24	n/a	n/a
GetAlarmBase			128	128	128	128	128	128
GetAlarm			216	216	216	216	216	216
SetRelAlarm			772	772	772	772	772	772
SetAbsAlarm			820	820	820	820	820	820
CancelAlarm			204	204	204	204	204	204
InitCounter			132	132	132	132	132	132
GetCounterValue			164	164	164	164	164	164
GetScheduleTableStatus		34	112	156	156	112	156	156

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
NextScheduleTable		34	156	388	388	156	388	408
StartScheduleTable		34	228	344	344	228	344	344
StopScheduleTable		34	156	196	196	156	196	196
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	16	16	16
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		12	12	12	12	12	12
ScheduleTable expiry point	Final		76	76	76	76	76	76
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	52	52	52	52	52	52
Process container	Non-Yielding	33	36	36	36	36	36	36
osek_tick_alarm	<default>		172	172	172	172	172	172
	KL	2	68	68	68	68	68	68
osek_incr_counter			60	60	60	60	60	60
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			308	308	308	308	308	308
ShutdownOS	NoHook	12	108	108	108	108	108	108
	Hook	13	132	132	132	132	132	132
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			60	60	60	60	60	60
StopCOM			32	32	32	32	32	32
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	164	164	164	312	312	312
	CCCB	15	312	312	312	312	312	312
GetMessageResource			88	88	88	88	88	88
ReleaseMessageResource			80	80	80	80	80	80
GetMessageStatus			76	76	76	76	76	76
SendMessage	SW CCCA	1, 14	216	216	216	400	400	400
	SW CCCB	1, 15	376	376	376	400	400	400
	NS CCCA	14	216	216	216	400	400	400
	NS CCCB	15	376	376	376	400	400	400
	KL CCCA	2, 14	116	116	116	324	324	324
	KL CCCB	2, 15	300	300	300	324	324	324
main_dispatch	NoHook	12	232	232	292	232	232	292
	Hook	13	280	280	340	280	280	340
sub_dispatch	B1LF	19	48	48	48	48	48	48
	B1HI	20	164	164	164	164	164	164
	B1HF	21	172	172	172	172	172	172
	B2LI	22	n/a	140	180	n/a	140	180
	B2LF	23	n/a	144	184	n/a	144	184
	B2HI	24	n/a	440	528	n/a	440	528
	B2HF	25	n/a	448	536	n/a	448	536
	E1HI	26	n/a	n/a	n/a	576	576	664

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
	E1HF	27	n/a	n/a	n/a	584	584	672
	E2HI	28	n/a	n/a	n/a	n/a	n/a	664
	E2HF	29	n/a	n/a	n/a	n/a	n/a	672
ErrorHook support		16	80	80	80	80	80	80
	ServiceID	17	88	88	88	88	88	88
	Parameters	18	108	108	108	108	108	108
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	300	472	520	308	508	576
	NS		244	424	488	260	452	552
	KL	2	108	284	352	124	312	412
ChainTaskset	SWL	1, 8	228	420	484	236	448	540
	SWH	1, 9	296	500	556	304	528	604
	NSL	8	228	420	484	236	448	540
	NSH	9	288	492	556	296	520	604
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			128	128	128	128	128	128
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			128	128	128	128	128	128
TestSubTaskset			152	152	152	152	152	152
TestEquivalentTaskset			148	148	148	148	148	148
TickSchedule	SW	1	364	332	332	332	332	332
	NS		312	280	280	280	280	280
	KL	2	244	220	220	220	220	220
AdvanceSchedule	SW	1	340	300	300	300	300	300
	NS		288	248	248	248	248	248
	KL	2	224	176	176	176	176	176
StartSchedule			200	200	200	200	200	200
StopSchedule			172	172	172	172	172	172
GetScheduleStatus			232	232	232	232	232	232
GetScheduleValue			168	168	168	168	168	168
GetScheduleNext			20	20	20	20	20	20
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			12	12	12	12	12	12
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			12	12	12	12	12	12
TestArrivalpointWritable			56	56	56	56	56	56
GetExecutionTime			8	8	8	8	8	8
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			8	8	8	8	8	8
GetStackOffset			20	20	20	20	20	20

Timing

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	308	400	448	316	408	488
	NS		252	352	400	260	360	440
	KL	2	116	212	264	124	220	300
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	20	20	20	20	20	20
ChainTask	SWL	1, 8	236	332	380	244	340	420
	SWH	1, 9	280	372	420	288	380	460
	NSL	8	236	332	380	244	340	420
	NSH	9	272	364	412	280	372	452
Schedule			216	216	256	216	216	256
GetTaskID			44	44	44	44	44	44
GetTaskState			172	172	172	196	196	196
EnableAllInterrupts			32	32	32	32	32	32
DisableAllInterrupts			36	36	36	36	36	36
ResumeAllInterrupts			60	60	60	60	60	60
SuspendAllInterrupts			84	84	84	84	84	84
ResumeOSInterrupts			96	96	96	96	96	96
SuspendOSInterrupts			128	128	128	128	128	128
GetResource	Task	7	32	32	40	32	32	40
	Combined	6	184	184	184	184	184	184
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	236	236	236	236	236	236
	Combined	6	452	452	452	452	452	452
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	256	256	360
	NS		n/a	n/a	n/a	176	176	312
	NS1i	10	n/a	n/a	n/a	128	n/a	n/a
	KL	2	n/a	n/a	n/a	76	76	188
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	100	100	100
GetEvent			n/a	n/a	n/a	20	20	20
WaitEvent	<default>		n/a	n/a	n/a	524	524	828
	fp	11	n/a	n/a	n/a	564	564	928
	1i	10	n/a	n/a	n/a	196	n/a	n/a
GetAlarmBase			128	128	128	128	128	128
GetAlarm			216	216	216	216	216	216
SetRelAlarm			772	772	772	772	772	772
SetAbsAlarm			820	820	820	820	820	820
CancelAlarm			204	204	204	204	204	204
InitCounter			132	132	132	132	132	132
GetCounterValue			164	164	164	164	164	164
GetScheduleTableStatus		34	112	156	156	112	156	156

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
NextScheduleTable		34	156	388	408	156	388	408
StartScheduleTable		34	228	344	344	228	344	344
StopScheduleTable		34	156	196	196	156	196	196
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	16	16	16
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		12	12	12	12	12	12
ScheduleTable expiry point	Final		76	76	76	76	76	76
GetISRID		4	56	56	56	56	56	56
Process container	Yielding	32	52	52	52	52	52	52
Process container	Non-Yielding	33	36	36	36	36	36	36
osek_tick_alarm	<default>		172	172	172	172	172	172
	KL	2	68	68	68	68	68	68
osek_incr_counter			60	60	60	60	60	60
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			360	360	360	360	360	360
ShutdownOS	NoHook	12	108	108	108	108	108	108
	Hook	13	132	132	132	132	132	132
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			60	60	60	60	60	60
StopCOM			32	32	32	32	32	32
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	164	164	164	312	312	312
	CCCB	15	312	312	312	312	312	312
GetMessageResource			88	88	88	88	88	88
ReleaseMessageResource			80	80	80	80	80	80
GetMessageStatus			76	76	76	76	76	76
SendMessage	SW CCCA	1, 14	216	216	216	400	400	400
	SW CCCB	1, 15	376	376	376	400	400	400
	NS CCCA	14	216	216	216	400	400	400
	NS CCCB	15	376	376	376	400	400	400
	KL CCCA	2, 14	116	116	116	324	324	324
	KL CCCB	2, 15	300	300	300	324	324	324
main_dispatch	NoHook	12	248	248	308	248	248	308
	Hook	13	300	300	360	300	300	360
sub_dispatch	B1LF	19	36	36	36	36	36	36
	B1HI	20	136	136	136	136	136	136
	B1HF	21	144	144	144	144	144	144
	B2LI	22	n/a	84	128	n/a	84	128
	B2LF	23	n/a	88	132	n/a	88	132
	B2HI	24	n/a	332	420	n/a	332	420
	B2HF	25	n/a	340	428	n/a	340	428
	E1HI	26	n/a	n/a	n/a	556	556	644

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
	E1HF	27	n/a	n/a	n/a	564	564	652
	E2HI	28	n/a	n/a	n/a	n/a	n/a	644
	E2HF	29	n/a	n/a	n/a	n/a	n/a	652
ErrorHook support		16	80	80	80	80	80	80
	ServiceID	17	88	88	88	88	88	88
	Parameters	18	108	108	108	108	108	108
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	136	136	136	136	136	136
Timing_termination		4	156	156	156	156	156	156
ActivateTaskset	SW	1	300	472	520	308	508	576
	NS		244	424	488	260	452	552
	KL	2	108	284	352	124	312	412
ChainTaskset	SWL	1, 8	228	420	484	236	448	540
	SWH	1, 9	296	500	556	304	528	604
	NSL	8	228	420	484	236	448	540
	NSH	9	288	492	556	296	520	604
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			128	128	128	128	128	128
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			128	128	128	128	128	128
TestSubTaskset			152	152	152	152	152	152
TestEquivalentTaskset			148	148	148	148	148	148
TickSchedule	SW	1	364	332	332	332	332	332
	NS		312	280	280	280	280	280
	KL	2	244	220	220	220	220	220
AdvanceSchedule	SW	1	340	300	300	300	300	300
	NS		288	248	248	248	248	248
	KL	2	224	176	176	176	176	176
StartSchedule			200	200	200	200	200	200
StopSchedule			172	172	172	172	172	172
GetScheduleStatus			232	232	232	232	232	232
GetScheduleValue			168	168	168	168	168	168
GetScheduleNext			20	20	20	20	20	20
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			12	12	12	12	12	12
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			12	12	12	12	12	12
TestArrivalpointWritable			56	56	56	56	56	56
GetExecutionTime			208	208	208	208	208	208
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			20	20	20	20	20	20
GetStackOffset			20	20	20	20	20	20

Extended

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	464	564	608	472	572	644
	NS		560	660	704	568	668	740
	KL	2	312	412	456	320	420	492
TerminateTask	LExt	3	204	204	204	204	204	204
	H	5	256	256	256	256	256	256
ChainTask	SWL	1, 8	520	628	672	528	636	716
	SWH	1, 9	572	668	712	580	676	748
	NSL	8	636	744	788	644	752	832
	NSH	9	684	780	824	692	788	860
Schedule			452	452	492	452	452	492
GetTaskID			68	68	68	68	68	68
GetTaskState			424	424	424	432	432	432
EnableAllInterrupts			56	56	56	56	56	56
DisableAllInterrupts			60	60	60	60	60	60
ResumeAllInterrupts			152	152	152	152	152	152
SuspendAllInterrupts			108	108	108	108	108	108
ResumeOSInterrupts			188	188	188	188	188	188
SuspendOSInterrupts			152	152	152	152	152	152
GetResource	Task	7	728	728	664	728	728	664
	Combined	6	736	736	736	736	736	736
	CLEx	3	632	632	632	632	632	632
ReleaseResource	Task	7	644	644	644	644	644	644
	Combined	6	876	876	876	876	876	876
	CLEx	3	600	600	600	600	600	600
SetEvent	SW	1	n/a	n/a	n/a	528	528	652
	NS		n/a	n/a	n/a	616	616	740
	NS1i	10	n/a	n/a	n/a	432	n/a	n/a
	KL	2	n/a	n/a	n/a	392	392	516
	KL1i	2, 10	n/a	n/a	n/a	340	n/a	n/a
ClearEvent			n/a	n/a	n/a	316	316	316
GetEvent			n/a	n/a	n/a	264	264	264
WaitEvent	<default>		n/a	n/a	n/a	712	712	996
	fp	11	n/a	n/a	n/a	756	756	1096
	1i	10	n/a	n/a	n/a	392	n/a	n/a
GetAlarmBase			360	360	360	360	360	360
GetAlarm			340	340	340	340	340	340
SetRelAlarm			992	992	992	992	992	992
SetAbsAlarm			1008	1008	1008	1008	1008	1008
CancelAlarm			320	320	320	320	320	320
InitCounter			408	408	408	408	408	408
GetCounterValue			384	384	384	384	384	384
GetScheduleTableStatus		34	140	184	184	140	184	184

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
NextScheduleTable		34	180	412	432	180	412	432
StartScheduleTable		34	252	368	368	252	368	368
StopScheduleTable		34	180	220	220	180	220	220
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	16	16	16
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		12	12	12	12	12	12
ScheduleTable expiry point	Final		76	76	76	76	76	76
GetISRID		4	80	80	80	80	80	80
Process container	Yielding	32	52	52	52	52	52	52
Process container	Non-Yielding	33	36	36	36	36	36	36
osek_tick_alarm	<default>		268	268	268	268	268	268
	KL	2	68	68	68	68	68	68
osek_incr_counter			60	60	60	60	60	60
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			392	392	392	392	392	392
ShutdownOS	NoHook	12	116	116	116	116	116	116
	Hook	13	140	140	140	140	140	140
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			92	92	92	92	92	92
StopCOM			68	68	68	68	68	68
ReadFlag			52	52	52	52	52	52
ResetFlag			52	52	52	52	52	52
ReceiveMessage	CCCA	14	316	316	316	476	476	476
	CCCB	15	476	476	476	476	476	476
GetMessageResource			160	160	160	160	160	160
ReleaseMessageResource			160	160	160	160	160	160
GetMessageStatus			184	184	184	184	184	184
SendMessage	SW CCCA	1, 14	396	396	396	592	592	592
	SW CCCB	1, 15	568	568	568	592	592	592
	NS CCCA	14	396	396	396	592	592	592
	NS CCCB	15	568	568	568	592	592	592
	KL CCCA	2, 14	292	292	292	488	488	488
	KL CCCB	2, 15	464	464	464	488	488	488
main_dispatch	NoHook	12	248	248	308	248	248	308
	Hook	13	300	300	360	300	300	360
sub_dispatch	B1LF	19	36	36	36	36	36	36
	B1HI	20	136	136	136	136	136	136
	B1HF	21	144	144	144	144	144	144
	B2LI	22	n/a	84	128	n/a	84	128
	B2LF	23	n/a	88	132	n/a	88	132
	B2HI	24	n/a	332	420	n/a	332	420
	B2HF	25	n/a	340	428	n/a	340	428
	E1HI	26	n/a	n/a	n/a	556	556	644

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	E1HF	27	n/a	n/a	n/a	564	564	652
	E2HI	28	n/a	n/a	n/a	n/a	n/a	644
	E2HF	29	n/a	n/a	n/a	n/a	n/a	652
ErrorHook support		16	272	272	272	272	272	272
	ServiceID	17	280	280	280	280	280	280
	Parameters	18	308	308	308	308	308	308
validity_checks		3	56	56	56	56	56	56
Timing_dispatch		4	136	136	136	136	136	136
Timing_termination		4	156	156	156	156	156	156
ActivateTaskset	SW	1	616	688	752	624	712	816
	NS		704	780	836	712	816	908
	KL	2	460	556	592	472	584	652
ChainTaskset	SWL	1, 8	676	744	808	680	764	860
	SWH	1, 9	752	840	900	756	860	956
	NSL	8	788	872	944	792	888	1000
	NSH	9	900	984	1052	904	1004	1116
GetTasksetRef			228	228	228	228	228	228
MergeTaskset			556	556	556	556	556	556
AssignTaskset			344	344	344	344	344	344
RemoveTaskset			556	556	556	556	556	556
TestSubTaskset			580	580	580	580	580	580
TestEquivalentTaskset			576	576	576	576	576	576
TickSchedule	SW	1	584	544	544	544	544	544
	NS		672	656	656	656	656	656
	KL	2	460	404	404	404	404	404
AdvanceSchedule	SW	1	600	556	556	556	556	556
	NS		688	672	672	672	672	672
	KL	2	488	420	420	420	420	420
StartSchedule			476	476	476	476	476	476
StopSchedule			392	392	392	392	392	392
GetScheduleStatus			448	448	448	448	448	448
GetScheduleValue			372	372	372	372	372	372
GetScheduleNext			160	160	160	160	160	160
SetScheduleNext			324	324	324	324	324	324
GetArrivalpointDelay			240	240	240	240	240	240
SetArrivalpointDelay			268	268	268	268	268	268
GetArrivalpointTasksetRef			236	236	236	236	236	236
GetArrivalpointNext			240	240	240	240	240	240
SetArrivalpointNext			372	372	372	372	372	372
TestArrivalpointWritable			272	272	272	272	272	272
GetExecutionTime			328	328	328	328	328	328
GetLargestExecutionTime			184	184	184	184	184	184
ResetLargestExecutionTime			168	168	168	168	168	168
GetStackOffset			20	20	20	20	20	20

Notes

Number	Note
1	Linked only if upward activations are allowed
2	Linked only if API is called within ISR
3	Present only in Extended OS status
4	Present only in Timing or Extended OS status
5	Linked only if there are heavyweight tasks in the system
6	Linked only if Resource is used by both tasks and ISRs
7	Linked only if Resource is used only by tasks
8	Linked only if Chaining task is Lightweight
9	Linked only if Chaining task is Heavyweight
10	Linked only if Idle task is the only extended task in the system
11	Linked only if calling Extended task uses floating-point
12	Linked only if neither Pre- nor Post-TaskHook is used
13	Linked only if Pre- or Post-TaskHook is used
14	Linked only if there are no flags, message queues, or message resources in the system, and COM status is not requested.
15	Linked only if there are any flags, message queues, or message resources in the system, or COM status is requested.
16	Linked only if `USEGETSERVICEID = FALSE` and `USEPARAMETERACCESS = FALSE`
17	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = FALSE`
18	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = TRUE`
19	Linked only for basic, single-activation, lightweight, floating-point tasks
20	Linked only for basic, single-activation, heavyweight, integer tasks
21	Linked only for basic, single-activation, heavyweight, floating-point tasks
22	Linked only for basic, multiple-activation, lightweight, integer tasks
23	Linked only for basic, multiple-activation, lightweight, floating-point tasks
24	Linked only for basic, multiple-activation, heavyweight, integer tasks
25	Linked only for basic, multiple-activation, heavyweight, floating-point tasks
26	Linked only for extended, unique priority, integer tasks
27	Linked only for extended, unique priority, floating-point tasks
28	Linked only for extended, shared priority, integer tasks
29	Linked only for extended, shared priority, floating-point tasks
30	Implemented as a macro, so no code is linked
31	Not required on some targets
32	Container for 2 process functions, not highest priority
33	Container for 2 process functions, highest or APPMODE or ISR
34	code varies with number of schedule tables; example uses 2 schedule tables

4.2.4 Reserved Hardware Resources

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	138	186	216	140	170	212
	NS	122	170	200	124	158	204
	KL	56	106	136	58	90	136
TerminateTask	LExt	0	0	0	0	0	0
	H	156	156	156	160	154	154
ChainTask	SWL	256	320	386	302	330	410
	SWH	318	378	444	362	386	468
	NSL	256	322	382	302	330	408
	NSH	314	378	440	360	384	464
Schedule	SW	114	112	124	114	114	126
GetTaskID		28	28	26	28	28	26
GetTaskState		104	102	104	108	108	108
EnableAllInterrupts		16	16	16	16	16	16
DisableAllInterrupts		20	22	22	20	20	18
ResumeAllInterrupts		24	24	24	26	24	24
SuspendAllInterrupts		30	30	30	30	30	30
ResumeOSInterrupts		24	24	24	24	24	24
SuspendOSInterrupts		30	30	30	32	32	30
GetResource	Task	38	36	38	40	38	38
	Combined	94	92	94	92	94	92
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	110	110	108	108	108	108
	Combined	184	184	184	184	184	184
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	128	128	132

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
	NS	n/a	n/a	n/a	114	112	118
	KL	n/a	n/a	n/a	50	52	48
ClearEvent		n/a	n/a	n/a	60	60	60
GetEvent		n/a	n/a	n/a	22	22	24
WaitEvent	<default>	n/a	n/a	n/a	426	426	466
	fp	n/a	n/a	n/a	432	436	470
GetAlarmBase		96	96	98	96	96	96
GetAlarm		112	112	116	114	116	114
SetRelAlarm		146	146	146	148	146	144
SetAbsAlarm		140	140	138	136	140	138
CancelAlarm		100	100	100	100	100	102
InitCounter		92	90	92	90	92	88
GetCounterValue		94	94	94	94	94	94
osek_tick_alarm	<default>	98	98	98	100	98	98
	KL	32	32	32	32	32	32
osek_incr_counter		12	12	12	12	12	12
GetActiveApplicationMode		8	8	8	8	8	8
StartOS		734	666	670	666	666	672
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	68	68	68	68	68	68
InitCOM		12	10	10	10	10	10
CloseCOM		10	10	10	10	10	10
StartCOM		36	36	34	72	76	74
StopCOM		18	18	18	20	18	20
ReadFlag		n/a	n/a	n/a	16	16	18
ResetFlag		n/a	n/a	n/a	16	16	16
ReceiveMessage		98	96	98	224	226	226
GetMessageResource		n/a	n/a	n/a	102	100	102
ReleaseMessageResource		n/a	n/a	n/a	180	180	176
GetMessageStatus		n/a	n/a	n/a	46	42	44
SendMessage	SW	258	304	336	392	422	464
	NS	238	286	318	372	406	452
	KL	110	162	192	242	282	322
ActivateTaskset	SW	114	398	458	116	396	474
	NS	98	384	430	102	382	452
	KL	32	316	362	32	312	380
	SW2	116	398	460	114	398	474
	NS2	98	384	430	102	380	452
	KL2	32	316	364	32	312	382
ChainTaskset	SWL	242	540	626	284	560	664
	SWH	304	608	692	350	624	724
	NSL	242	538	622	282	558	664
	NSH	306	606	690	346	622	726
GetTasksetRef		20	24	24	22	20	22
MergeTaskset		84	84	84	84	84	82

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
AssignTaskset		14	14	14	12	14	14
RemoveTaskset		78	78	78	80	80	80
TestSubTaskset		90	88	88	90	90	88
TestEquivalentTaskset		86	88	88	88	88	88
TickSchedule	SW	174	494	540	206	506	566
	NS	160	482	530	192	492	554
	KL	96	416	462	126	422	486
	SW2	172	494	540	206	492	558
	NS2	160	482	530	192	478	546
	KL2	96	416	462	126	410	478
AdvanceSchedule	SW	154	468	516	180	478	542
	NS	134	452	498	166	464	526
	KL	78	396	440	110	406	470
	SW2	154	470	516	180	466	534
	NS2	136	454	498	168	450	518
	KL2	80	398	440	110	392	460
StartSchedule		112	112	108	114	114	112
StopSchedule		102	104	102	104	104	104
GetScheduleStatus		116	118	116	116	116	116
GetScheduleValue		108	106	106	108	106	106
GetScheduleNext		20	20	22	22	22	22
SetScheduleNext		18	18	20	18	16	16
GetArrivalpointDelay		18	18	18	18	20	18
SetArrivalpointDelay		12	12	14	12	12	12
GetArrivalpointTasksetRef		14	14	14	14	14	14
GetArrivalpointNext		14	14	12	14	14	16
SetArrivalpointNext		14	14	14	14	12	12
TestArrivalpointWritable		26	26	28	26	28	26
GetExecutionTime		12	12	12	12	14	12
GetLargestExecutionTime		20	20	18	18	16	18
ResetLargestExecutionTime		16	16	16	16	16	14
GetStackOffset		14	14	16	14	14	14

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	140	186	216	138	172	212
	NS	122	174	200	122	156	206
	KL	54	108	134	56	90	136

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
TerminateTask	LExt	0	0	0	0	0	0
	H	330	330	330	330	326	330
ChainTask	SWL	484	546	608	528	558	648
	SWH	546	598	664	586	612	706
	NSL	488	546	614	528	554	652
	NSH	540	598	660	580	604	702
Schedule	SW	118	112	128	114	116	124
GetTaskID		26	28	26	28	26	28
GetTaskState		104	102	102	108	106	110
EnableAllInterrupts		16	16	16	16	16	16
DisableAllInterrupts		20	22	20	20	20	18
ResumeAllInterrupts		24	24	24	26	24	24
SuspendAllInterrupts		30	30	30	30	30	30
ResumeOSInterrupts		24	24	24	24	24	24
SuspendOSInterrupts		32	30	30	32	32	30
GetResource	Task	38	36	38	38	38	38
	Combined	92	94	94	92	90	92
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	110	112	112	112	110	112
	Combined	186	186	186	186	186	186
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	126	128	132
	NS	n/a	n/a	n/a	110	114	118
	KL	n/a	n/a	n/a	48	48	48
ClearEvent		n/a	n/a	n/a	60	60	62
GetEvent		n/a	n/a	n/a	24	24	22
WaitEvent	<default>	n/a	n/a	n/a	606	592	644
	fp	n/a	n/a	n/a	616	600	654
GetAlarmBase		96	96	96	92	98	94
GetAlarm		114	114	114	116	118	114
SetRelAlarm		146	144	148	144	150	144
SetAbsAlarm		138	140	140	138	138	136
CancelAlarm		102	102	100	100	100	102
InitCounter		92	92	90	94	92	90
GetCounterValue		96	98	94	94	94	94
osek_tick_alarm	<default>	100	98	100	100	96	98
	KL	30	32	32	32	34	32
osek_incr_counter		12	12	12	12	12	12
GetActiveApplicationMode		8	8	8	8	8	8
StartOS		1690	1688	1688	1694	1688	1688
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	70	68	70	68	68	68
InitCOM		10	10	12	10	10	10
CloseCOM		10	10	10	10	10	10
StartCOM		38	36	36	74	72	74

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
StopCOM		18	18	18	20	20	18
ReadFlag		n/a	n/a	n/a	18	18	16
ResetFlag		n/a	n/a	n/a	16	16	16
ReceiveMessage		96	98	98	224	226	226
GetMessageResource		n/a	n/a	n/a	98	102	102
ReleaseMessageResource		n/a	n/a	n/a	180	182	180
GetMessageStatus		n/a	n/a	n/a	42	42	44
SendMessage	SW	258	306	334	386	422	460
	NS	242	296	318	372	404	454
	KL	110	162	190	246	282	320
ActivateTaskset	SW	118	398	456	114	398	476
	NS	98	386	434	102	380	454
	KL	32	316	368	34	314	386
	SW2	116	396	456	116	398	476
	NS2	98	386	434	104	380	456
	KL2	30	316	368	34	314	386
ChainTaskset	SWL	468	764	854	510	790	910
	SWH	528	828	912	570	848	964
	NSL	466	766	854	510	784	908
	NSH	524	822	910	568	844	964
GetTasksetRef		22	20	22	20	22	20
MergeTaskset		84	84	86	84	84	84
AssignTaskset		14	16	14	14	14	14
RemoveTaskset		78	80	78	78	80	82
TestSubTaskset		90	88	90	90	90	90
TestEquivalentTaskset		86	86	86	88	86	86
TickSchedule	SW	174	490	546	208	502	570
	NS	158	480	528	196	490	558
	KL	92	412	462	126	422	492
	SW2	174	490	546	208	490	562
	NS2	160	480	530	196	478	550
	KL2	92	410	462	126	410	484
AdvanceSchedule	SW	150	466	520	182	480	544
	NS	136	450	504	168	460	528
	KL	80	394	446	110	404	472
	SW2	150	468	520	182	468	538
	NS2	134	452	504	166	450	520
	KL2	80	394	446	110	390	464
StartSchedule		110	116	110	110	112	114
StopSchedule		102	104	106	104	104	104
GetScheduleStatus		114	116	116	116	116	116
GetScheduleValue		104	108	104	108	106	108
GetScheduleNext		22	22	22	22	22	22
SetScheduleNext		18	18	18	20	16	18
GetArrivalpointDelay		20	18	18	20	20	20

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
SetArrivalpointDelay		14	14	14	12	12	12
GetArrivalpointTasksetRef		12	12	14	14	14	16
GetArrivalpointNext		14	14	12	16	14	14
SetArrivalpointNext		14	14	14	12	14	12
TestArrivalpointWritable		26	26	26	26	26	26
GetExecutionTime		124	126	126	128	126	128
GetLargestExecutionTime		30	30	30	30	28	30
ResetLargestExecutionTime		26	26	28	26	26	26
GetStackOffset		14	14	14	14	14	14

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	360	404	434	354	390	428
	NS	408	452	486	404	436	480
	KL	292	336	368	286	324	358
TerminateTask	LExt	406	408	408	404	406	402
	H	444	444	444	442	446	442
ChainTask	SWL	794	862	930	834	876	958
	SWH	846	900	972	890	920	994
	NSL	852	916	992	892	934	1012
	NSH	902	962	1032	942	974	1056
Schedule	SW	172	174	188	174	176	186
GetTaskID		34	36	34	34	34	36
GetTaskState		356	352	354	350	354	354
EnableAllInterrupts		24	22	22	24	24	24
DisableAllInterrupts		26	26	26	28	26	28
ResumeAllInterrupts		46	44	44	44	44	46
SuspendAllInterrupts		38	36	36	38	36	38
ResumeOSInterrupts		46	44	44	44	44	44
SuspendOSInterrupts		38	38	38	36	38	38
GetResource	Task	602	620	358	640	634	400
	Combined	336	336	334	368	372	378
	CLEx	350	346	348	386	386	388
ReleaseResource	Task	318	316	316	356	356	358
	Combined	364	364	364	404	402	408
	CLEx	308	306	308	346	348	342
SetEvent	SW	n/a	n/a	n/a	376	376	380
	NS	n/a	n/a	n/a	402	402	408

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
	KL	n/a	n/a	n/a	318	318	320
ClearEvent		n/a	n/a	n/a	134	132	134
GetEvent		n/a	n/a	n/a	272	274	274
WaitEvent	<default>	n/a	n/a	n/a	730	722	762
	fp	n/a	n/a	n/a	734	734	770
GetAlarmBase		258	258	270	254	254	272
GetAlarm		252	250	262	248	254	266
SetRelAlarm		302	302	312	298	298	316
SetAbsAlarm		288	286	298	282	282	300
CancelAlarm		236	234	246	230	234	248
InitCounter		370	370	388	374	374	400
GetCounterValue		238	236	234	234	238	238
osek_tick_alarm	<default>	128	128	128	130	126	130
	KL	32	30	32	30	32	32
osek_incr_counter		10	8	8	10	10	10
GetActiveApplicationMode		8	6	6	8	8	8
StartOS		1750	1744	1976	1750	1746	1748
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	72	72	70	72	72	70
InitCOM		10	10	10	10	10	10
CloseCOM		10	10	10	10	10	10
StartCOM		48	50	50	84	84	84
StopCOM		28	28	28	28	28	28
ReadFlag		n/a	n/a	n/a	32	30	32
ResetFlag		n/a	n/a	n/a	30	32	30
ReceiveMessage		210	208	210	324	330	328
GetMessageResource		n/a	n/a	n/a	506	510	512
ReleaseMessageResource		n/a	n/a	n/a	496	498	490
GetMessageStatus		n/a	n/a	n/a	132	136	136
SendMessage	SW	580	622	654	696	742	774
	NS	626	670	704	748	788	828
	KL	464	508	540	580	620	656
ActivateTaskset	SW	462	760	812	468	748	826
	NS	506	804	858	506	796	868
	KL	388	696	752	394	682	768
	SW2	462	760	812	468	748	826
	NS2	506	804	858	508	796	870
	KL2	388	696	752	392	682	766
ChainTaskset	SWL	922	1234	1322	964	1242	1364
	SWH	974	1284	1378	1022	1296	1412
	NSL	968	1284	1374	1012	1292	1410
	NSH	1028	1334	1434	1076	1352	1470
GetTasksetRef		260	258	256	252	256	256
MergeTaskset		200	196	194	196	194	196
AssignTaskset		98	100	100	100	102	100

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
RemoveTaskset		192	196	190	192	194	190
TestSubTaskset		206	202	204	202	204	204
TestEquivalentTaskset		200	202	198	200	202	200
TickSchedule	SW	238	910	964	600	920	996
	NS	280	946	1000	640	962	1034
	KL	170	838	890	530	854	930
	SW2	238	910	966	600	894	976
	NS2	278	946	1002	640	938	1016
	KL2	172	838	894	530	830	910
AdvanceSchedule	SW	232	900	950	590	910	988
	NS	276	944	996	636	954	1034
	KL	170	836	888	526	844	924
	SW2	232	900	950	590	884	968
	NS2	276	946	998	636	928	1014
	KL2	170	834	890	526	818	904
StartSchedule		194	190	192	196	192	192
StopSchedule		156	158	156	156	158	156
GetScheduleStatus		164	166	168	164	166	162
GetScheduleValue		158	160	160	162	162	160
GetScheduleNext		56	56	54	54	56	56
SetScheduleNext		100	102	100	100	100	100
GetArrivalpointDelay		72	72	72	72	76	74
SetArrivalpointDelay		80	82	80	82	82	82
GetArrivalpointTasksetRef		56	58	56	56	56	58
GetArrivalpointNext		56	54	56	58	58	58
SetArrivalpointNext		106	106	106	104	106	104
TestArrivalpointWritable		68	66	68	68	68	68
GetExecutionTime		156	156	154	154	154	156
GetLargestExecutionTime		244	242	244	242	244	242
ResetLargestExecutionTime		240	238	240	230	236	234
GetStackOffset		18	18	16	16	16	16

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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	49	49	49	49	49	49
	Cat 2	114	168	166	168	166	166

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	49	49	49	49	49	49
	Cat 2	228	276	280	280	276	278

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	49	49	49	49	49	49
	Cat 2	226	278	280	278	278	274

4.3.4 Task Switching Times

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

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

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

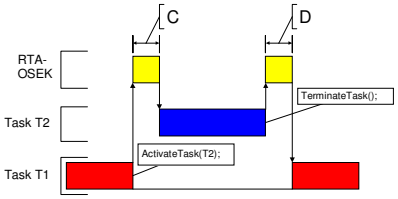

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

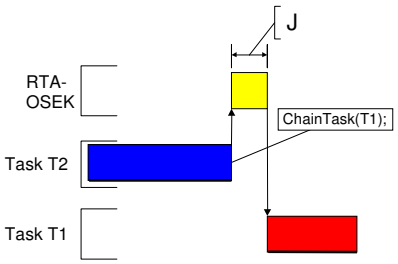

Figure 2: Task Chaining

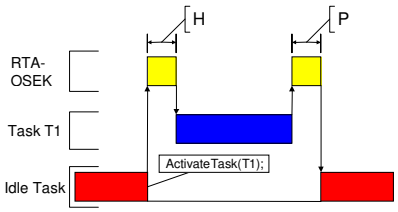

Figure 3: Task Activation from Idle Task

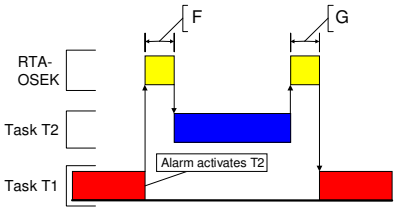

Figure 4: Task Activation from an Alarm

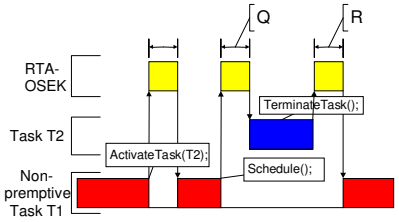

Figure 5: Non-Premptive Task Calls Schedule()

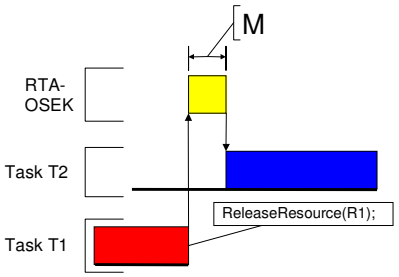

Figure 6: Blocked Task Activated by ReleaseResource()

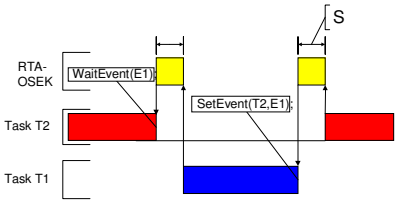

Figure 7: Waiting Task Activated by SetEvent()

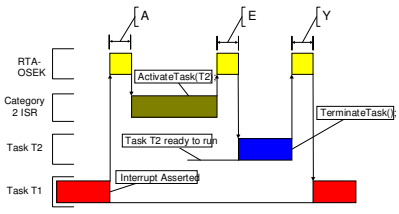

Figure 8: Category 2 ISR Activates a Higher Priority Task

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Normal termination	Light, Basic	100	142	166	102	140	164
Figure 1: D	Heavy, Basic/Extended	154	192	216	206	200	224
ChainTask	Light, Basic	202	274	334	210	266	348
Figure 2: J	Heavy, Basic/Extended	456	556	644	510	560	668
Pre-emption	Light, Basic	188	258	328	188	252	344
Figure 1: C	Heavy, Basic/Extended	256	318	390	292	328	412
From idle task	Light, Basic	192	254	328	194	254	344
Figure 3: H	Heavy, Basic/Extended	262	318	390	302	330	416
Triggered by alarm	Light, Basic	304	364	436	304	362	454
Figure 4: F	Heavy, Basic/Extended	366	424	496	408	436	520
Schedule	Light, Basic	158	172	228	156	170	226
Figure 5: Q	Heavy, Basic/Extended	224	234	290	264	260	314
Release resource	Light, Basic	192	206	248	192	202	248
Figure 6: M	Heavy, Basic/Extended	258	268	312	298	294	338
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	460	460	566
From category 2 ISR	Light, Basic	134	204	246	188	200	246
Figure 8: E	Heavy, Basic/Extended	198	264	306	294	288	332

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	280	300	332	282	300	328
Figure 1: D	Heavy, Basic/Extended	330	344	370	356	352	376
ChainTask	Light, Basic	434	492	562	438	494	580
Figure 2: J	Heavy, Basic/Extended	862	924	1022	890	932	1052
Pre-emption	Light, Basic	302	354	428	298	350	450
Figure 1: C	Heavy, Basic/Extended	362	420	490	398	430	528
From idle task	Light, Basic	304	356	428	300	352	452
Figure 3: H	Heavy, Basic/Extended	364	420	492	400	432	530
Triggered by alarm	Light, Basic	412	460	536	408	456	560
Figure 4: F	Heavy, Basic/Extended	470	526	600	510	536	638
Schedule	Light, Basic	268	270	328	266	270	332
Figure 5: Q	Heavy, Basic/Extended	328	336	390	366	362	426
Release resource	Light, Basic	300	302	350	298	302	354
Figure 6: M	Heavy, Basic/Extended	358	368	412	400	396	448

Configuration / Events / Shared Task Priorities / Multiple Task Activations	Task Attributes	Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	542	536	664
From category 2 ISR	Light, Basic	418	480	528	480	478	532
Figure 8: E	Heavy, Basic/Extended	478	546	590	580	572	628

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations	Task Attributes	Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Normal termination	Light, Basic	400	426	450	400	424	450
Figure 1: D	Heavy, Basic/Extended	444	458	482	472	470	490
ChainTask	Light, Basic	740	810	876	740	806	886
Figure 2: J	Heavy, Basic/Extended	1276	1346	1442	1306	1358	1452
Pre-emption	Light, Basic	504	556	634	500	552	638
Figure 1: C	Heavy, Basic/Extended	562	616	698	602	638	722
From idle task	Light, Basic	504	556	634	500	554	640
Figure 3: H	Heavy, Basic/Extended	562	616	698	604	638	722
Triggered by alarm	Light, Basic	638	692	766	636	688	776
Figure 4: F	Heavy, Basic/Extended	698	752	830	740	774	860
Schedule	Light, Basic	300	310	374	310	316	372
Figure 5: Q	Heavy, Basic/Extended	360	372	438	412	410	466
Release resource	Light, Basic	482	492	542	522	528	572
Figure 6: M	Heavy, Basic/Extended	540	554	604	624	624	668
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	790	792	906
From category 2 ISR	Light, Basic	448	510	564	504	512	558
Figure 8: E	Heavy, Basic/Extended	508	572	626	608	608	652

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		160	160	160	160	160	160
BCC1 lightweight, floating-point		176	176	176	176	176	176
BCC1 heavyweight, integer		272	272	272	272	272	272
BCC1 heavyweight, floating-point		272	272	272	272	272	272
BCC2 lightweight, integer		n/a	176	176	n/a	176	176
BCC2 lightweight, floating-point		n/a	176	176	n/a	176	176
BCC2 heavyweight, integer		n/a	288	288	n/a	288	288
BCC2 heavyweight, floating-point		n/a	288	288	n/a	288	288
ECC1 heavyweight, integer		n/a	n/a	n/a	304	304	304
ECC1 heavyweight, floating-point		n/a	n/a	n/a	304	304	304
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	304
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	304
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		160	160	176	160	160	176
BCC1 lightweight, floating-point		176	176	192	176	176	192
BCC1 heavyweight, integer		272	272	288	272	272	288
BCC1 heavyweight, floating-point		272	272	288	272	272	288
BCC2 lightweight, integer		n/a	176	192	n/a	176	192
BCC2 lightweight, floating-point		n/a	176	192	n/a	176	192
BCC2 heavyweight, integer		n/a	288	304	n/a	288	304
BCC2 heavyweight, floating-point		n/a	288	304	n/a	288	304
ECC1 heavyweight, integer		n/a	n/a	n/a	304	304	320
ECC1 heavyweight, floating-point		n/a	n/a	n/a	304	304	320
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	320
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	320

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		192	192	192	192	192	192
BCC1 lightweight, floating-point		208	208	208	208	208	208
BCC1 heavyweight, integer		320	320	320	320	320	320

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 heavyweight, floating-point		320	320	320	320	320	320
BCC2 lightweight, integer		n/a	208	208	n/a	208	208
BCC2 lightweight, floating-point		n/a	208	208	n/a	208	208
BCC2 heavyweight, integer		n/a	320	320	n/a	320	320
BCC2 heavyweight, floating-point		n/a	320	320	n/a	320	320
ECC1 heavyweight, integer		n/a	n/a	n/a	368	368	368
ECC1 heavyweight, floating-point		n/a	n/a	n/a	368	368	368
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	368
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	368
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		192	192	192	192	192	192
BCC1 lightweight, floating-point		208	208	208	208	208	208
BCC1 heavyweight, integer		320	320	320	320	320	320
BCC1 heavyweight, floating-point		320	320	320	320	320	320
BCC2 lightweight, integer		n/a	208	208	n/a	208	208
BCC2 lightweight, floating-point		n/a	208	208	n/a	208	208
BCC2 heavyweight, integer		n/a	320	320	n/a	320	320
BCC2 heavyweight, floating-point		n/a	320	320	n/a	320	320
ECC1 heavyweight, integer		n/a	n/a	n/a	368	368	368
ECC1 heavyweight, floating-point		n/a	n/a	n/a	368	368	368
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	368
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	368

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		192	192	192	192	192	192
BCC1 lightweight, floating-point		208	208	208	208	208	208
BCC1 heavyweight, integer		320	320	320	320	320	320
BCC1 heavyweight, floating-point		320	320	320	320	320	320
BCC2 lightweight, integer		n/a	208	208	n/a	208	208
BCC2 lightweight, floating-point		n/a	208	208	n/a	208	208
BCC2 heavyweight, integer		n/a	320	320	n/a	320	320
BCC2 heavyweight, floating-point		n/a	320	320	n/a	320	320
ECC1 heavyweight, integer		n/a	n/a	n/a	384	384	384
ECC1 heavyweight, floating-point		n/a	n/a	n/a	384	384	384

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	384
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	384
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		192	192	192	192	192	192
BCC1 lightweight, floating-point		208	208	208	208	208	208
BCC1 heavyweight, integer		320	320	320	320	320	320
BCC1 heavyweight, floating-point		320	320	320	320	320	320
BCC2 lightweight, integer		n/a	208	208	n/a	208	208
BCC2 lightweight, floating-point		n/a	208	208	n/a	208	208
BCC2 heavyweight, integer		n/a	320	320	n/a	320	320
BCC2 heavyweight, floating-point		n/a	320	320	n/a	320	320
ECC1 heavyweight, integer		n/a	n/a	n/a	384	384	384
ECC1 heavyweight, floating-point		n/a	n/a	n/a	384	384	384
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	384
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	384

5 Compatibility with Pre-v5 Kernels

5.1 Updating the Application Version

5.1.1 OS Version

To convert an existing v3.x OIL configuration file to v5.0, load the file into the v5 RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.0. When the OIL configuration file is saved it will then use the v5.0 format and the v5.0 kernel libraries. This process can be reversed to move back to earlier kernel versions.

5.1.2 Namespace changes

The RTA-OSEK kernel now uses names with the prefix os_ for kernel specific external objects and section names. This means that linker control files will need to be updated to ensure correct section placement and initialization. The names used for the API functions and variables have not changed.

Support

For product support, please contact your local ETAS representative.

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