RTA-OSEK

Binding Manual: XCV2/Tasking

Contact Details

ETAS Group www.etasgroup.com	
ETAS GmbH 70469 Stuttgart, Germany Tel.:+49 711 89661-0 Fax:+49 711 89661-300 sales.de@etas.com	**ETAS Inc.** Ann Arbor, MI 48103, USA Tel.: +1 888 ETAS INC Fax: +1 734 997 9449 sales.us@etas.com
ETAS K.K. Yokohama 220-6217, Japan Tel.: +81 45 222-0900 Fax: +81 45 222-0956 sales.jp@etas.com	**ETAS S.A.S.** 94588 Rungis Cedex, France Tel.: +33 (1) 56 70 00 50 Fax: +33 (1) 56 70 00 51 sales.fr@etas.com
ETAS Korea Co. Ltd. Seoul 137-889, Korea Tel.: +82 2 5747-016 Fax: +82 2 5747-120 sales.kr@etas.com	**ETAS Ltd.** Burton-upon-Trent Staffordshire DE14 2WQ, UK Tel.: +44 1283 54 65 12 Fax: +44 1283 54 87 67 sales.uk@etas.com
ETAS (Shanghai) Co., Ltd. Shanghai 200120, P.R. China Tel.: +86 21 5037 2220 Fax: +86 21 5037 2221 sales.cn@etas.com	**ETAS Automotive India Pvt. Ltd.** Bangalore 560 068, India Tel.: +91 80 4191 2585 Fax: +91 80 4191 2586 sales.in@etas.com

Copyright Notice

Version: M00091-002

Disclaimer

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

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

Trademarks

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

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

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

Contents

1 About this Guide

This guide provides target-specific information for the XCV2/Tasking port of LiveDevices' RTA-OSEK. It supplements the more general information in the *RTA-OSEK User Guide*.

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

1.1 Who Should Read this Guide?

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

1.2 Conventions

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

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

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

2 Toolchain Issues

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

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

2.1 Memory Model

The RTA-OSEK runtime libraries have been built for the small memory model. The OS code in the "`OS_WRAPPER_CLASS`" class must be located within a single code segment.

All externally visible (API) OS functions can be called from user programs in any code segment.

DPP3 (Data page Pointer Register 3) should be fixed at address 0xC000.

2.2 Compiler

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

Vendor	Tasking
Compiler	Tasking VX-toolset for C166/ST10
Version	v2.3r2 Build 592

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

Option	Description
`-Mn`	Near memory model
`-C<cpu>`	Selects target CPU

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

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

Option	Description
`-Mn`	Near memory model
`-C<cpu>`	Selects target CPU

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

Option	Description
-Ob	Uninitialized memory is not zeroed

To support the use of multiple CPU configurations the environment variable CPU_TYPE should be set up to match the desired CPU target (e.g. xc161cj-16ff).

The Tasking supplied C startup code, `cstart.c`, can be used unmodified. However the included header file `cstart.h` comes with some of the on chip peripherals disabled the user has to modify SYSCON3 register settings to enable on chip peripherals.

As the Tasking startup code performs the EINIT instruction the register access control prohibits the modification of some registers including the SYSCON3 register. The security level must be changed before these registers can be updated, this is demonstrated in the example application in the function `init_target()`. For further details on register access control please refer to the Infineon User's Manual of the relevant microprocessor.

2.3 Assembler

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

Vendor	Tasking
Assembler	Tasking VX-toolset for C166: Assembler
Version	v2.3r2 Build 268

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

Option	Description
-C<cpu>	Selects target CPU

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

2.4 Linker/Locator

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

Option	Description
-C<cpu>	Selects target CPU

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
os_pird	ROM	RTA-OSEK initialization data
os_vectbl	ROM	Vector table if generated by RTA-OSEK GUI
os_pir	RAM	RTA-OSEK initialized data; initialized during StartOS()
os_pur	RAM	RTA-OSEK uninitialized data; zeroed during StartOS()
os_text	ROM	RTA-OSEK code
os_data2	RAM	RTA-OSEK data; must be initialized during C-startup
os_trace_ram	RAM	RTA-TRACE uninitialized data; must be zeroed during C-startup
OS_WRAPPER_CLASS	ROM	RTA-OSEK interrupt wrappers

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

C Library Functions	Description
setjmp	ISO C library function
longjmp	ISO C library function
cstart.c	Startup code with automatic initialization
pnearp32	C library function
icall	C library function
cpnnw	C library function
vectab	C library function
mul	C library function
_init	C library function
_dcti	C library function
_Exit	C library function

This port of RTA-OSEK is built for the `c166cnlib`, `c166rtn.lib` runtime libraries. The compiler flag -Mn specifies that data pointers are 16-bits and function pointers are 24-bits.

2.4.1 Section Placement

The RTA-OSEK code section OS_WAPPER_CLASS must all be linked within a single code segment. The following linker script fragment forces the Tasking linker do this (as demonstrated in the example application linker script).

```
section_layout ::code
{
        group ( ordered, contiguous, page )
        {
                select *OS_WRAPPER_CLASS*";
        }
}
```

2.5 Debugger

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

ORTI Compatible Debuggers
TASKING VX Debugger
Lauterbach TRACE32

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for XCV2/Tasking. You can find out more about configuring interrupts for RTA-OSEK in the *RTA-OSEK User Guide*.

3.1.1 Interrupt Levels

In RTA-OSEK interrupts are allocated an Interrupt Priority Level (IPL). This is a processor independent abstraction of the interrupt priorities that are available on the target hardware. You can find out more about IPLs in the *RTA-OSEK User Guide*. The hardware interrupt controller is explained in the *Infineon XC16x User Manual*.

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

IPL Value	ILVL Value	Description
0	0000	User level
1-13	0001–1101	Category 1 and 2 interrupts
14-15	1110–1111	Category 1 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
0x02,0x04,0x06,0x0A	Category 1, Priority 15 only
For XC161: 0x60 to 0x7F	Category 1 only
For XC22XX: 0x70 to 0x7F	Category 1only
For XC2785X: 0x70 to 0x7F	Category 1 only

The valid base addresses for the vector table are:

Base Address	Notes
0x xx 0000	For XC16x (xx is defined in Reg VECSEG the segment where the vector table is located to)

3.1.3 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Tasking C compiler can generate appropriate interrupt handling code for a C function decorated with the `__interrupt` 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.src`.

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 (VVVV represents the 4 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
0xVVVV	`_os_wrapper_VVVV`
e.g. `0x004C`	`_os_wrapper_004C`

The vector table generated by the RTA-OSEK GUI locates itself at address 0x8, leaving the reset vector at address 0x0 unaffected, and does not scale.

Even when the RTA-OSEK GUI does not generate a vector table, suitable directives are included in `osgen.src` to allow the Tasking linker/locator to build a vector table that includes correct entries for the declared Category 2 interrupts. We recommend using this method of building a vector table.

The Vector base is programmable using VECSEG register, which gives the segment number (high byte of 24-bit address) for the start of the vector table. Thus the vector table can be located on any 64 Kb boundary. The reset configuration allows initial setting on one of 5 locations for most memory configurations.

VECSC is a 2-bit filed in the CPUCON1 register which sets the size of vector table entries. The default size is 2 words (same as C166), but it can be set to 4, 8, or 16 words. The RTA-OSEK component generates vector tables that use 2 word entries if the user wishes to change the vector size an alternative user generated vector table must be provided.

3.2 Register Settings

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

PSW Field	Required Value	Notes
IEN	1	Permits interrupts

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

PSW Field	Notes
ILVL	Used to control IPL
IEN	Must not be cleared
MULIP	Must not be directly manipulated by user code
VECSEG	Specifies the location of the vector table

3.3 Stack Usage

3.3.1 Number of Stacks

Two stacks are used. The System stack is indexed 0 and the User stack is indexed 1.

The first argument to `StackFaultHook` is 0 or 1. This indicates the stack on which the excess was observed.

`osStackOffsetType` is a structure of two scalars, representing the number of bytes on each stack. This is shown in Code Example 3:2.

```
typedef unsigned short  osStackBytes0;
typedef unsigned short  osStackBytes1;
typedef struct {
        osStackBytes0   sys;
        osStackBytes1   usr;
} StackOffsetType;
```

Code Example 3:2 - osStackOffsetType()

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

Stack	API Max Usage (Bytes)
System stack	8
User stack (C stack)	28

Timing

Stack	API Max Usage (Bytes)
System stack	8
User stack (C stack)	28

Extended

Stack	API Max Usage (Bytes)
System stack	8
User stack (C stack)	28

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.

RTA-OSEK needs the following Stack symbols (stack bottom and stack top) to be assigned to the assembler predefined symbols in linker script file.

Stack symbol	Assemble pre defined symbol
_os_user_stack_top	__lc_ub_user_stack
_os_system_stack_top	__lc_ub_system_stack
_os_user_stack_bottom	__lc_ue_user_stack
_os_system_stack_bottom	__lc_ue_system_stack

e.g.

```
section_layout ::huge
{
        "_os_system_stack_top" :=
"__lc_ub_system_stack";
```

```
        "_os_system_stack_bottom" :=
"__lc_ue_system_stack";
}

section_layout ::near (direction = high_to_low)
{
        group ( run_addr = [0x7800.. 0x7FF0],
ordered ) stack "user_stack"( size = 2032 );
        "_os_user_stack_top" :=
"__lc_ub_user_stack";
        "_os_user_stack_bottom" :=
"__lc_ue_user_stack";
}
```

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	**XC161**
Clock speed (MHz)	20
Code memory	Internal FLash
Read-only data memory	Internal ROM
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	27	27	27	27	27	27
	ROM	111	111	115	191	191	195
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	45	45	45	45	45	45
	ROM	177	177	181	257	257	261
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	53	53	53	53	53	53
	ROM	203	203	207	283	283	287
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	24	24	24	24	24	24
BCC1 Heavyweight task	RAM	2	2	2	2	2	2
	ROM	26	26	26	26	26	26
BCC2 task	RAM	n/a	4	6	n/a	4	6
	ROM	n/a	32	36	n/a	32	36
ECC1, Integer task	RAM	n/a	n/a	n/a	28	28	28
	ROM	n/a	n/a	n/a	42	42	42
ECC1, floating-point task	RAM	n/a	n/a	n/a	30	30	30
	ROM	n/a	n/a	n/a	42	42	42
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	30
	ROM	n/a	n/a	n/a	n/a	n/a	46
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	32
	ROM	n/a	n/a	n/a	n/a	n/a	46
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	34	34	34	34	34	34
Category 2 ISR, floating-point	RAM	2	2	2	2	2	2
	ROM	46	46	46	46	46	46
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	10	10	10	10	10	10
	ROM	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	104	104	104	104	104	104
Message	RAM	11	11	11	31	31	31
	ROM	12	12	12	30	30	30
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	6	0	6	6
ScheduleTable	RAM	10	10	10	10	10	10
	ROM	82	82	82	82	82	82
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	8	8	8	8	8	8
Arrivalpoint (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8
Schedule	RAM	12	12	12	12	12	12
	ROM	26	26	26	26	26	26
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	10	10	10	10	10	10
	ROM	32	32	32	32	32	32
BCC1 Heavyweight task	RAM	12	12	12	12	12	12
	ROM	34	34	34	34	34	34
BCC2 task	RAM	n/a	14	16	n/a	14	16
	ROM	n/a	40	44	n/a	40	44
ECC1, Integer task	RAM	n/a	n/a	n/a	38	38	38
	ROM	n/a	n/a	n/a	50	50	50
ECC1, floating-point task	RAM	n/a	n/a	n/a	40	40	40
	ROM	n/a	n/a	n/a	50	50	50
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	40

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	n/a	n/a	n/a	n/a	n/a	54
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	42
	ROM	n/a	n/a	n/a	n/a	n/a	54
Category 2 ISR	RAM	10	10	10	10	10	10
	ROM	60	60	60	60	60	60
Category 2 ISR, floating-point	RAM	12	12	12	12	12	12
	ROM	68	68	68	68	68	68
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	10	10	10	10	10	10
	ROM	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	104	104	104	104	104	104
Message	RAM	11	11	11	31	31	31
	ROM	12	12	12	30	30	30
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	6	0	6	6
ScheduleTable	RAM	10	10	10	10	10	10
	ROM	82	82	82	82	82	82
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	8	8	8	8	8	8
Arrivalpoint (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8
Schedule	RAM	12	12	12	12	12	12

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	26	26	26	26	26	26
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	12	12	12	12	12	12
	ROM	36	36	36	36	36	36
BCC1 Heavyweight task	RAM	14	14	14	14	14	14
	ROM	36	36	36	36	36	36
BCC2 task	RAM	n/a	16	18	n/a	16	18
	ROM	n/a	42	46	n/a	42	46
ECC1, Integer task	RAM	n/a	n/a	n/a	40	40	40
	ROM	n/a	n/a	n/a	52	52	52
ECC1, floating-point task	RAM	n/a	n/a	n/a	42	42	42
	ROM	n/a	n/a	n/a	52	52	52
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	42
	ROM	n/a	n/a	n/a	n/a	n/a	56
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	44
	ROM	n/a	n/a	n/a	n/a	n/a	56
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	64	64	64	64	64	64
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	72	72	72	72	72	72
Resource	RAM	4	4	4	4	4	4
	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	4	4	4	4	4	4
	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	
Alarm	RAM	10	10	10	10	10	10
	ROM	32	32	32	32	32	32
Counter	RAM	4	4	4	4	4	4
	ROM	106	106	106	106	106	106
Message	RAM	11	11	11	31	31	31
	ROM	14	14	14	32	32	32
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	4	4	4	4	4	4
	ROM	14	14	14	14	14	14
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	6	0	6	6
ScheduleTable	RAM	10	10	10	10	10	10
	ROM	82	82	82	82	82	82
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	14	14	14	14	14	14
Arrivalpoint (writable)	RAM	14	14	14	14	14	14
	ROM	14	14	14	14	14	14
Schedule	RAM	16	16	16	16	16	16
	ROM	32	32	32	32	32	32
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.
Hook	Pre- and Post- Task hooks are used.
KL	API is called from OS level.
KL1i	API is called from OS level, idle task is only ECC task.
KL2	Activated taskset has one BCC2 task.
LExt	Used for lightweight termination in Extended Status.
ServiceID	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
Parameters	`ErrorHook` uses `GetServiceID` and `GetServiceParameters`.
NoHook	Pre- and/or Post- Task hooks are not used.
NS	No context switch is possible.
NS1i	No context switch is possible, idle task is only ECC task.
NS2	Activated taskset has one BCC2 task.
NSH	Chain from heavyweight task, not to higher priority.
NSL	Chain from lightweight task, not to higher priority.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.

Variant	Description
Task	Resource is used only by tasks.

Standard

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	236	356	456	244	372	496
	NS		196	316	416	204	324	456
	KL	2	124	244	344	132	252	384
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	32	32	32	32	32	32
ChainTask	SWL	1, 8	188	308	408	196	316	448
	SWH	1, 9	228	344	456	248	356	496
	NSL	8	188	308	408	196	316	448
	NSH	9	220	336	448	240	348	488
Schedule			148	148	212	148	148	212
GetTaskID			48	48	48	48	48	48
GetTaskState			120	120	120	148	148	148
EnableAllInterrupts			36	36	36	36	36	36
DisableAllInterrupts			44	44	44	44	44	44
ResumeAllInterrupts			58	58	58	58	58	58
SuspendAllInterrupts			64	64	64	64	64	64
ResumeOSInterrupts			58	58	58	58	58	58
SuspendOSInterrupts			94	94	94	94	94	94
GetResource	Task	7	40	40	52	40	40	52
	Combined	6	130	130	130	130	130	130
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	120	120	120	120	120	120
	Combined	6	272	272	272	272	272	272
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	216	216	376
	NS		n/a	n/a	n/a	168	168	328
	NS1i	10	n/a	n/a	n/a	88	n/a	n/a
	KL	2	n/a	n/a	n/a	116	116	276
	KL1i	2, 10	n/a	n/a	n/a	36	n/a	n/a
ClearEvent			n/a	n/a	n/a	76	76	76

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetEvent			n/a	n/a	n/a	28	28	28
WaitEvent	<default>		n/a	n/a	n/a	516	516	980
	fp	11	n/a	n/a	n/a	592	592	1140
	1i	10	n/a	n/a	n/a	36	n/a	n/a
GetAlarmBase			92	92	92	92	92	92
GetAlarm			232	232	232	232	232	232
SetRelAlarm			1524	1524	1524	1524	1524	1524
SetAbsAlarm			1892	1892	1892	1892	1892	1892
CancelAlarm			172	172	172	172	172	172
InitCounter			136	136	136	136	136	136
GetCounterValue			172	172	172	172	172	172
GetScheduleTableStatus		34	78	96	96	78	96	96
NextScheduleTable		34	100	196	196	100	196	196
StartScheduleTable		34	126	176	176	126	176	176
StopScheduleTable		34	86	104	104	86	104	104
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	10	10	10
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		28	28	28	28	28	28
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	24	24	24	24	24	24
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		160	160	160	160	160	160
	KL	2	108	108	108	108	108	108
osek_incr_counter			128	128	128	128	128	128
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			196	196	196	196	196	196
ShutdownOS	NoHook	12	62	62	62	62	62	62
	Hook	13	70	70	70	70	70	70
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			40	40	40	40	40	40
StopCOM			28	28	28	28	28	28
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	112	112	112	380	380	380

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	380	380	380	380	380	380
GetMessageResource			64	64	64	64	64	64
ReleaseMessageResource			52	52	52	52	52	52
GetMessageStatus			100	100	100	100	100	100
SendMessage	SW CCCA	1, 14	200	200	200	492	492	492
	SW CCCB	1, 15	464	464	464	492	492	492
	NS CCCA	14	200	200	200	492	492	492
	NS CCCB	15	464	464	464	492	492	492
	KL CCCA	2, 14	148	148	148	440	440	440
	KL CCCB	2, 15	412	412	412	440	440	440
main_dispatch	NoHook	12	232	232	304	232	232	304
	Hook	13	300	300	368	300	300	368
sub_dispatch	B1LF	19	64	64	64	64	64	64
	B1HI	20	168	168	168	168	168	168
	B1HF	21	184	184	184	184	184	184
	B2LI	22	n/a	136	200	n/a	136	200
	B2LF	23	n/a	152	216	n/a	152	216
	B2HI	24	n/a	462	652	n/a	462	652
	B2HF	25	n/a	478	668	n/a	478	668
	E1HI	26	n/a	n/a	n/a	614	614	782
	E1HF	27	n/a	n/a	n/a	630	630	798
	E2HI	28	n/a	n/a	n/a	n/a	n/a	782
	E2HF	29	n/a	n/a	n/a	n/a	n/a	798
ErrorHook support		16	52	52	52	52	52	52
	ServiceID	17	64	64	64	64	64	64
	Parameters	18	116	116	116	116	116	116
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	232	420	540	252	460	624
	NS		192	380	492	212	420	576
	KL	2	120	308	420	140	348	504
ChainTaskset	SWL	1, 8	196	408	520	208	436	576
	SWH	1, 9	244	472	588	256	500	656
	NSL	8	196	408	520	208	436	576
	NSH	9	236	464	580	248	492	648
GetTasksetRef			12	12	12	12	12	12

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
MergeTaskset			72	72	72	72	72	72
AssignTaskset			12	12	12	12	12	12
RemoveTaskset			76	76	76	76	76	76
TestSubTaskset			92	92	92	92	92	92
TestEquivalentTaskset			84	84	84	84	84	84
TickSchedule	SW	1	460	412	412	412	412	412
	NS		412	348	348	348	348	348
	KL	2	352	288	288	288	288	288
AdvanceSchedule	SW	1	428	352	352	352	352	352
	NS		380	304	304	304	304	304
	KL	2	328	244	244	244	244	244
StartSchedule			180	180	180	180	180	180
StopSchedule			136	136	136	136	136	136
GetScheduleStatus			212	212	212	212	212	212
GetScheduleValue			188	188	188	188	188	188
GetScheduleNext			16	16	16	16	16	16
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			80	80	80	80	80	80
SetArrivalpointDelay			76	76	76	76	76	76
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			20	20	20	20	20	20
SetArrivalpointNext			16	16	16	16	16	16
TestArrivalpointWritable			48	48	48	48	48	48
GetExecutionTime			12	12	12	12	12	12
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			8	8	8	8	8	8
GetStackOffset			56	56	56	56	56	56

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	236	356	456	244	372	496
	NS		196	316	416	204	324	456
	KL	2	124	244	344	132	252	384
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	32	32	32	32	32	32
ChainTask	SWL	1, 8	188	308	408	196	316	448
	SWH	1, 9	228	344	456	248	356	496
	NSL	8	188	308	408	196	316	448
	NSH	9	220	336	448	240	348	488
Schedule			180	180	244	180	180	244
GetTaskID			48	48	48	48	48	48
GetTaskState			120	120	120	148	148	148
EnableAllInterrupts			36	36	36	36	36	36
DisableAllInterrupts			44	44	44	44	44	44
ResumeAllInterrupts			58	58	58	58	58	58
SuspendAllInterrupts			64	64	64	64	64	64
ResumeOSInterrupts			58	58	58	58	58	58
SuspendOSInterrupts			94	94	94	94	94	94
GetResource	Task	7	40	40	52	40	40	52
	Combined	6	130	130	130	130	130	130
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	152	152	152	152	152	152
	Combined	6	336	336	336	336	336	336
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	216	216	376
	NS		n/a	n/a	n/a	168	168	328
	NS1i	10	n/a	n/a	n/a	88	n/a	n/a
	KL	2	n/a	n/a	n/a	116	116	276
	KL1i	2, 10	n/a	n/a	n/a	36	n/a	n/a
ClearEvent			n/a	n/a	n/a	76	76	76
GetEvent			n/a	n/a	n/a	28	28	28
WaitEvent	<default>		n/a	n/a	n/a	688	688	1148
	fp	11	n/a	n/a	n/a	756	756	1304

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	1i	10	n/a	n/a	n/a	172	n/a	n/a
GetAlarmBase			92	92	92	92	92	92
GetAlarm			232	232	232	232	232	232
SetRelAlarm			1524	1524	1524	1524	1524	1524
SetAbsAlarm			1892	1892	1892	1892	1892	1892
CancelAlarm			172	172	172	172	172	172
InitCounter			136	136	136	136	136	136
GetCounterValue			172	172	172	172	172	172
GetScheduleTableStatus		34	78	96	96	78	96	96
NextScheduleTable		34	100	196	196	100	196	196
StartScheduleTable		34	126	176	176	126	176	176
StopScheduleTable		34	86	104	104	86	104	104
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	10	10	10
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		28	28	28	28	28	28
GetISRID		4	28	28	28	28	28	28
Process container	Yielding	32	24	24	24	24	24	24
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		160	160	160	160	160	160
	KL	2	108	108	108	108	108	108
osek_incr_counter			128	128	128	128	128	128
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			312	312	312	312	312	312
ShutdownOS	NoHook	12	62	62	62	62	62	62
	Hook	13	70	70	70	70	70	70
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			40	40	40	40	40	40
StopCOM			28	28	28	28	28	28
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	112	112	112	380	380	380
	CCCB	15	380	380	380	380	380	380
GetMessageResource			64	64	64	64	64	64
ReleaseMessageResource			52	52	52	52	52	52

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			100	100	100	100	100	100
SendMessage	SW CCCA	1, 14	200	200	200	492	492	492
	SW CCCB	1, 15	464	464	464	492	492	492
	NS CCCA	14	200	200	200	492	492	492
	NS CCCB	15	464	464	464	492	492	492
	KL CCCA	2, 14	148	148	148	440	440	440
	KL CCCB	2, 15	412	412	412	440	440	440
main_dispatch	NoHook	12	388	388	456	388	388	456
	Hook	13	448	448	524	448	448	524
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	144	144	144	144	144	144
	B1HF	21	160	160	160	160	160	160
	B2LI	22	n/a	76	140	n/a	76	140
	B2LF	23	n/a	88	152	n/a	88	152
	B2HI	24	n/a	400	588	n/a	400	588
	B2HF	25	n/a	416	604	n/a	416	604
	E1HI	26	n/a	n/a	n/a	636	636	804
	E1HF	27	n/a	n/a	n/a	652	652	820
	E2HI	28	n/a	n/a	n/a	n/a	n/a	804
	E2HF	29	n/a	n/a	n/a	n/a	n/a	820
ErrorHook support		16	52	52	52	52	52	52
	ServiceID	17	64	64	64	64	64	64
	Parameters	18	116	116	116	116	116	116
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	152	152	152	152	152	152
Timing_termination		4	180	180	180	180	180	180
ActivateTaskset	SW	1	232	420	540	252	460	624
	NS		192	380	492	212	420	576
	KL	2	120	308	420	140	348	504
ChainTaskset	SWL	1, 8	196	408	520	208	436	576
	SWH	1, 9	244	472	588	256	500	656
	NSL	8	196	408	520	208	436	576
	NSH	9	236	464	580	248	492	648
GetTasksetRef			12	12	12	12	12	12
MergeTaskset			72	72	72	72	72	72
AssignTaskset			12	12	12	12	12	12
RemoveTaskset			76	76	76	76	76	76

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			92	92	92	92	92	92
TestEquivalentTaskset			84	84	84	84	84	84
TickSchedule	SW	1	460	412	412	412	412	412
	NS		412	348	348	348	348	348
	KL	2	352	288	288	288	288	288
AdvanceSchedule	SW	1	428	352	352	352	352	352
	NS		380	304	304	304	304	304
	KL	2	328	244	244	244	244	244
StartSchedule			180	180	180	180	180	180
StopSchedule			136	136	136	136	136	136
GetScheduleStatus			212	212	212	212	212	212
GetScheduleValue			188	188	188	188	188	188
GetScheduleNext			16	16	16	16	16	16
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			80	80	80	80	80	80
SetArrivalpointDelay			76	76	76	76	76	76
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			20	20	20	20	20	20
SetArrivalpointNext			16	16	16	16	16	16
TestArrivalpointWritable			48	48	48	48	48	48
GetExecutionTime			196	196	196	196	196	196
GetLargestExecutionTime			88	88	88	88	88	88
ResetLargestExecutionTime			88	88	88	88	88	88
GetStackOffset			56	56	56	56	56	56

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	436	556	656	456	564	700
	NS		500	624	724	520	632	764
	KL	2	288	408	508	308	416	552
TerminateTask	LExt	3	200	200	200	200	200	200

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	256	256	256	256	256	256
ChainTask	SWL	1, 8	472	604	704	492	612	744
	SWH	1, 9	544	672	772	564	680	812
	NSL	8	556	688	788	580	700	828
	NSH	9	628	756	856	652	768	896
Schedule			424	424	488	424	424	488
GetTaskID			72	72	72	72	72	72
GetTaskState			452	452	452	452	452	452
EnableAllInterrupts			60	60	60	60	60	60
DisableAllInterrupts			76	76	76	76	76	76
ResumeAllInterrupts			172	172	172	172	172	172
SuspendAllInterrupts			96	96	96	96	96	96
ResumeOSInterrupts			172	172	172	172	172	172
SuspendOSInterrupts			126	126	126	126	126	126
GetResource	Task	7	652	652	612	652	652	612
	Combined	6	642	642	642	642	642	642
	CLEx	3	592	592	592	592	592	592
ReleaseResource	Task	7	600	600	600	600	600	600
	Combined	6	790	790	790	790	790	790
	CLEx	3	552	552	552	552	552	552
SetEvent	SW	1	n/a	n/a	n/a	560	560	724
	NS		n/a	n/a	n/a	628	628	792
	NS1i	10	n/a	n/a	n/a	440	n/a	n/a
	KL	2	n/a	n/a	n/a	412	412	576
	KL1i	2, 10	n/a	n/a	n/a	328	n/a	n/a
ClearEvent			n/a	n/a	n/a	252	252	252
GetEvent			n/a	n/a	n/a	336	336	336
WaitEvent	<default>		n/a	n/a	n/a	936	936	1376
	fp	11	n/a	n/a	n/a	1004	1004	1532
	1i	10	n/a	n/a	n/a	408	n/a	n/a
GetAlarmBase			376	376	376	376	376	376
GetAlarm			380	380	380	380	380	380
SetRelAlarm			1804	1804	1804	1804	1804	1804
SetAbsAlarm			2160	2160	2160	2160	2160	2160
CancelAlarm			316	316	316	316	316	316
InitCounter			500	500	500	500	500	500
GetCounterValue			472	472	472	472	472	472

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	94	112	112	94	112	112
NextScheduleTable		34	116	212	212	116	212	212
StartScheduleTable		34	142	192	192	142	192	192
StopScheduleTable		34	102	120	120	102	120	120
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	10	10	10
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		28	28	28	28	28	28
GetISRID		4	44	44	44	44	44	44
Process container	Yielding	32	24	24	24	24	24	24
Process container	Non-Yielding	33	12	12	12	12	12	12
osek_tick_alarm	<default>		268	268	268	268	268	268
	KL	2	108	108	108	108	108	108
osek_incr_counter			128	128	128	128	128	128
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			344	344	344	344	344	344
ShutdownOS	NoHook	12	78	78	78	78	78	78
	Hook	13	86	86	86	86	86	86
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			72	72	72	72	72	72
StopCOM			72	72	72	72	72	72
ReadFlag			48	48	48	48	48	48
ResetFlag			56	56	56	56	56	56
ReceiveMessage	CCCA	14	308	308	308	572	572	572
	CCCB	15	572	572	572	572	572	572
GetMessageResource			156	156	156	156	156	156
ReleaseMessageResource			156	156	156	156	156	156
GetMessageStatus			192	192	192	192	192	192
SendMessage	SW CCCA	1, 14	392	392	392	672	672	672
	SW CCCB	1, 15	644	644	644	672	672	672
	NS CCCA	14	392	392	392	672	672	672
	NS CCCB	15	644	644	644	672	672	672
	KL CCCA	2, 14	300	300	300	580	580	580
	KL CCCB	2, 15	552	552	552	580	580	580
main_dispatch	NoHook	12	388	388	456	388	388	456

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	448	448	524	448	448	524
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	148	148	148	148	148	148
	B1HF	21	164	164	164	164	164	164
	B2LI	22	n/a	76	140	n/a	76	140
	B2LF	23	n/a	88	152	n/a	88	152
	B2HI	24	n/a	404	592	n/a	404	592
	B2HF	25	n/a	420	608	n/a	420	608
	E1HI	26	n/a	n/a	n/a	644	644	812
	E1HF	27	n/a	n/a	n/a	660	660	828
	E2HI	28	n/a	n/a	n/a	n/a	n/a	812
	E2HF	29	n/a	n/a	n/a	n/a	n/a	828
ErrorHook support		16	192	192	192	192	192	192
	ServiceID	17	208	208	208	208	208	208
	Parameters	18	260	260	260	260	260	260
validity_checks		3	40	40	40	40	40	40
Timing_dispatch		4	152	152	152	152	152	152
Timing_termination		4	180	180	180	180	180	180
ActivateTaskset	SW	1	532	688	796	604	740	860
	NS		600	752	864	668	796	928
	KL	2	372	528	640	440	572	708
ChainTaskset	SWL	1, 8	592	746	858	648	770	902
	SWH	1, 9	684	836	952	740	860	1004
	NSL	8	684	862	974	760	886	1018
	NSH	9	768	928	1048	828	956	1092
GetTasksetRef			280	280	280	280	280	280
MergeTaskset			568	568	568	568	568	568
AssignTaskset			388	388	388	388	388	388
RemoveTaskset			572	572	572	572	572	572
TestSubTaskset			592	592	592	592	592	592
TestEquivalentTaskset			580	580	580	580	580	580
TickSchedule	SW	1	720	644	644	644	644	644
	NS		784	792	792	792	792	792
	KL	2	608	532	532	532	532	532
AdvanceSchedule	SW	1	740	636	636	636	636	636
	NS		812	808	808	808	808	808
	KL	2	636	524	524	524	524	524

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			528	528	528	528	528	528
StopSchedule			400	400	400	400	400	400
GetScheduleStatus			516	516	516	516	516	516
GetScheduleValue			504	504	504	504	504	504
GetScheduleNext			216	216	216	216	216	216
SetScheduleNext			380	380	380	380	380	380
GetArrivalpointDelay			440	440	440	440	440	440
SetArrivalpointDelay			468	468	468	468	468	468
GetArrivalpointTasksetRef			288	288	288	288	288	288
GetArrivalpointNext			296	296	296	296	296	296
SetArrivalpointNext			436	436	436	436	436	436
TestArrivalpointWritable			324	324	324	324	324	324
GetExecutionTime			284	284	284	284	284	284
GetLargestExecutionTime			400	400	400	400	400	400
ResetLargestExecutionTime			336	336	336	336	336	336
GetStackOffset			56	56	56	56	56	56

Notes

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

Number	Note
16	Linked only if `USEGETSERVICEID = FALSE` and `USEPARAMETERACCESS = FALSE`
17	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = FALSE`
18	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = TRUE`
19	Linked only for basic, single-activation, lightweight, floating-point tasks
20	Linked only for basic, single-activation, heavyweight, integer tasks
21	Linked only for basic, single-activation, heavyweight, floating-point tasks
22	Linked only for basic, multiple-activation, lightweight, integer tasks
23	Linked only for basic, multiple-activation, lightweight, floating-point tasks
24	Linked only for basic, multiple-activation, heavyweight, integer tasks
25	Linked only for basic, multiple-activation, heavyweight, floating-point tasks
26	Linked only for extended, unique priority, integer tasks
27	Linked only for extended, unique priority, floating-point tasks
28	Linked only for extended, shared priority, integer tasks
29	Linked only for extended, shared priority, floating-point tasks
30	Implemented as a macro, so no code is linked
31	Not required on some targets
32	Container for 2 process functions, not highest priority
33	Container for 2 process functions, highest or APPMODE or ISR
34	code varies with number of schedule tables; example uses 2 schedule tables

4.2.4 Reserved Hardware Resources

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

4.3 Performance

The collection of performance data for the XCV2/Tasking port of the RTA-OSEK Component was achieved using a timer running four times slower than the CPU clock speed. The figures in this section, therefore, have an uncertainty level of up to four CPU cycles. The actual times are between 0 and four cycles shorter than those reported in the remainder of this section.

The XC161CJ processor used for collecting timing data exhibits considerable variation in measured execution times of sections of code. This depends largely on memory configuration (particularly wait states used to access external memory), and there can also be an effect due to code alignment. In the test configuration, a variation of up to 8 cycles was observed for some APIs. All the times shown here are pessimistic measurements based on this observed variation.

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	107	161	217	108	165	225
	NS	93	149	202	96	148	209
	KL	51	103	159	49	104	166
TerminateTask	LExt	0	0	0	0	0	0
	H	185	185	185	187	187	191
ChainTask	SWL	289	354	461	406	459	559
	SWH	376	434	541	490	539	648
	NSL	289	354	461	402	459	559
	NSH	375	433	542	492	531	648
Schedule	SW	99	101	120	102	103	123
GetTaskID		32	35	34	37	36	35
GetTaskState		80	83	82	109	107	108
EnableAllInterrupts		37	39	37	39	39	38
DisableAllInterrupts		29	27	31	29	31	34
ResumeAllInterrupts		43	47	47	51	49	48
SuspendAllInterrupts		32	33	35	34	34	35
ResumeOSInterrupts		45	49	47	49	49	48
SuspendOSInterrupts		32	33	36	34	36	37
GetResource	Task	37	38	40	39	36	42
	Combined	78	78	82	82	84	83
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	70	73	72	75	77	78
	Combined	123	125	123	127	127	128
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	112	112	113
	NS	n/a	n/a	n/a	103	101	99

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	KL	n/a	n/a	n/a	70	68	69
ClearEvent		n/a	n/a	n/a	60	60	61
GetEvent		n/a	n/a	n/a	18	22	23
WaitEvent	<default>	n/a	n/a	n/a	519	526	577
	fp	n/a	n/a	n/a	540	545	597
GetAlarmBase		142	142	143	144	143	145
GetAlarm		95	99	98	97	97	97
SetRelAlarm		151	152	150	149	151	152
SetAbsAlarm		134	130	132	134	132	133
CancelAlarm		79	79	79	82	83	84
InitCounter		81	81	84	85	83	84
GetCounterValue		78	78	74	78	80	81
osek_tick_alarm	<default>	87	87	85	85	87	88
	KL	50	50	52	54	52	53
osek_incr_counter		17	17	16	18	15	18
GetActiveApplicationMode		4	4	4	4	3	4
StartOS		601	607	601	605	608	608
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	50	50	50	52	52	53
InitCOM		12	14	12	16	15	16
CloseCOM		12	14	12	16	15	16
StartCOM		43	40	42	133	133	134
StopCOM		29	28	27	31	29	30
ReadFlag		n/a	n/a	n/a	10	7	10
ResetFlag		n/a	n/a	n/a	9	6	9
ReceiveMessage		66	66	66	248	246	247
GetMessageResource		n/a	n/a	n/a	80	77	80
ReleaseMessageResource		n/a	n/a	n/a	128	129	129
GetMessageStatus		n/a	n/a	n/a	74	71	72
SendMessage	SW	210	264	317	383	437	498
	NS	200	256	307	374	425	486
	KL	121	171	227	295	350	412
ActivateTaskset	SW	74	444	511	79	445	497
	NS	64	429	465	67	417	484
	KL	28	393	432	32	387	453
	SW2	74	444	508	79	445	497

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NS2	64	429	465	67	417	484
	KL2	28	393	432	32	387	453
ChainTaskset	SWL	268	623	730	382	746	847
	SWH	361	732	828	477	834	947
	NSL	266	623	730	382	745	847
	NSH	363	718	825	476	835	946
GetTasksetRef		21	21	15	17	18	19
MergeTaskset		51	50	52	54	54	55
AssignTaskset		16	18	16	14	17	18
RemoveTaskset		52	53	52	54	52	53
TestSubTaskset		57	55	54	58	58	59
TestEquivalentTaskset		56	56	56	58	57	58
TickSchedule	SW	203	581	623	218	580	644
	NS	185	564	602	200	559	622
	KL	152	528	572	169	526	589
	SW2	202	580	622	217	574	640
	NS2	185	564	602	200	554	618
	KL2	152	528	567	169	521	585
AdvanceSchedule	SW	167	550	590	186	547	609
	NS	147	531	566	171	528	592
	KL	117	498	536	142	499	562
	SW2	167	550	587	186	542	605
	NS2	147	531	566	171	523	588
	KL2	116	498	536	142	494	559
StartSchedule		127	131	130	134	133	134
StopSchedule		109	111	109	113	113	114
GetScheduleStatus		88	89	88	91	91	92
GetScheduleValue		119	117	116	117	117	120
GetScheduleNext		17	17	20	19	18	19
SetScheduleNext		20	20	17	17	20	21
GetArrivalpointDelay		66	68	68	72	70	71
SetArrivalpointDelay		53	54	51	54	56	57
GetArrivalpointTasksetRef		16	14	18	16	15	16
GetArrivalpointNext		18	18	16	16	19	20
SetArrivalpointNext		16	16	18	18	17	18
TestArrivalpointWritable		22	22	20	20	23	24

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetExecutionTime		14	14	16	14	15	18
GetLargestExecutionTime		24	19	22	20	20	21
ResetLargestExecutionTime		14	14	14	16	17	18
GetStackOffset		20	18	20	20	21	22

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	110	164	217	112	166	222
	NS	98	151	207	98	151	212
	KL	51	101	158	52	108	163
TerminateTask	LExt	0	0	0	0	0	0
	H	417	414	417	425	424	421
ChainTask	SWL	538	613	718	670	721	818
	SWH	615	675	786	742	787	895
	NSL	538	613	723	670	721	826
	NSH	623	678	788	747	795	900
Schedule	SW	102	102	122	103	104	123
GetTaskID		36	35	39	36	35	35
GetTaskState		80	81	80	115	113	111
EnableAllInterrupts		39	38	39	39	38	38
DisableAllInterrupts		29	29	29	33	32	34
ResumeAllInterrupts		51	48	51	51	50	48
SuspendAllInterrupts		35	35	34	36	35	35
ResumeOSInterrupts		49	48	49	49	48	48
SuspendOSInterrupts		35	35	34	36	35	37
GetResource	Task	40	37	41	38	40	42
	Combined	80	79	80	84	81	83
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	75	74	75	74	73	78
	Combined	127	128	125	127	126	128
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetEvent	SW	n/a	n/a	n/a	113	113	110
	NS	n/a	n/a	n/a	104	103	101
	KL	n/a	n/a	n/a	72	72	69
ClearEvent		n/a	n/a	n/a	63	62	64
GetEvent		n/a	n/a	n/a	24	25	21
WaitEvent	<default>	n/a	n/a	n/a	750	752	795
	fp	n/a	n/a	n/a	768	765	821
GetAlarmBase		146	145	146	147	148	150
GetAlarm		97	98	96	99	98	99
SetRelAlarm		151	148	154	157	156	152
SetAbsAlarm		136	131	133	135	134	137
CancelAlarm		81	83	82	84	84	84
InitCounter		83	82	83	85	84	84
GetCounterValue		79	78	78	81	80	78
osek_tick_alarm	<default>	87	84	87	92	91	88
	KL	52	51	54	56	55	53
osek_incr_counter		18	16	17	17	18	18
GetActiveApplicationMode		4	4	5	5	4	4
StartOS		2132	2129	2130	2138	2138	2135
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	54	51	54	52	51	53
InitCOM		16	16	17	15	16	16
CloseCOM		16	16	17	15	16	16
StartCOM		47	42	46	138	137	133
StopCOM		31	28	31	31	30	30
ReadFlag		n/a	n/a	n/a	9	10	10
ResetFlag		n/a	n/a	n/a	8	9	9
ReceiveMessage		68	67	68	241	240	243
GetMessageResource		n/a	n/a	n/a	80	81	80
ReleaseMessageResource		n/a	n/a	n/a	132	132	131
GetMessageStatus		n/a	n/a	n/a	69	68	70
SendMessage	SW	213	265	317	386	439	488
	NS	203	258	312	365	417	483
	KL	123	170	225	298	354	406
ActivateTaskset	SW	76	442	484	80	429	526
	NS	64	430	494	67	432	486

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	KL	30	381	434	32	400	481
	SW2	76	442	481	80	429	526
	NS2	64	430	494	67	432	486
	KL2	30	381	435	32	400	482
ChainTaskset	SWL	519	890	1018	650	1008	1099
	SWH	600	975	1069	728	1071	1185
	NSL	527	889	1024	658	1015	1099
	NSH	601	967	1095	727	1077	1186
GetTasksetRef		22	19	20	20	24	18
MergeTaskset		53	54	54	54	53	57
AssignTaskset		18	16	19	19	20	18
RemoveTaskset		54	55	52	57	56	53
TestSubTaskset		59	60	59	59	58	59
TestEquivalentTaskset		57	56	57	59	59	57
TickSchedule	SW	202	567	616	223	598	667
	NS	189	550	600	204	579	650
	KL	155	516	571	172	546	617
	SW2	202	567	616	223	594	665
	NS2	189	550	600	204	575	645
	KL2	155	516	571	172	542	613
AdvanceSchedule	SW	169	534	586	186	561	639
	NS	151	519	574	174	551	622
	KL	120	486	540	144	516	587
	SW2	169	534	586	186	557	635
	NS2	151	519	574	174	547	618
	KL2	120	486	540	144	512	586
StartSchedule		133	131	128	132	130	134
StopSchedule		115	112	113	113	112	116
GetScheduleStatus		91	91	92	92	95	92
GetScheduleValue		117	114	115	120	119	118
GetScheduleNext		19	23	19	18	23	19
SetScheduleNext		23	19	23	20	19	23
GetArrivalpointDelay		68	69	70	69	68	69
SetArrivalpointDelay		53	53	54	55	52	54
GetArrivalpointTasksetRef		18	20	17	17	20	18
GetArrivalpointNext		20	18	21	19	18	20

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetArrivalpointNext		18	20	19	17	20	18
TestArrivalpointWritable		24	22	25	23	22	24
GetExecutionTime		117	117	117	124	121	123
GetLargestExecutionTime		79	78	81	81	80	82
ResetLargestExecutionTime		73	69	71	74	74	70
GetStackOffset		24	26	23	22	22	24

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	549	613	670	552	603	667
	NS	584	660	716	584	638	706
	KL	468	519	573	470	512	569
TerminateTask	LExt	484	478	486	487	491	490
	H	544	544	545	548	555	550
ChainTask	SWL	1090	1166	1277	1209	1254	1376
	SWH	1157	1217	1332	1279	1322	1436
	NSL	1125	1222	1338	1257	1292	1438
	NSH	1206	1268	1387	1324	1371	1488
Schedule	SW	186	180	204	188	187	205
GetTaskID		40	41	38	41	42	41
GetTaskState		548	564	557	556	565	570
EnableAllInterrupts		43	40	41	46	45	43
DisableAllInterrupts		37	34	33	36	37	35
ResumeAllInterrupts		61	64	65	60	64	67
SuspendAllInterrupts		39	39	39	39	40	40
ResumeOSInterrupts		61	64	62	63	62	65
SuspendOSInterrupts		40	39	39	39	40	40
GetResource	Task	858	862	424	946	947	516
	Combined	395	402	392	481	482	484
	CLEx	436	444	443	519	522	535
ReleaseResource	Task	381	383	384	467	468	476

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	Combined	397	388	392	484	485	483
	CLEx	380	383	385	457	462	474
SetEvent	SW	n/a	n/a	n/a	618	614	626
	NS	n/a	n/a	n/a	631	626	643
	KL	n/a	n/a	n/a	543	546	553
ClearEvent		n/a	n/a	n/a	108	109	106
GetEvent		n/a	n/a	n/a	489	491	502
WaitEvent	<default>	n/a	n/a	n/a	886	889	920
	fp	n/a	n/a	n/a	901	906	943
GetAlarmBase		395	404	431	400	398	433
GetAlarm		345	348	383	352	346	379
SetRelAlarm		463	462	493	460	459	493
SetAbsAlarm		414	413	455	422	418	452
CancelAlarm		328	330	365	334	331	365
InitCounter		607	603	609	606	609	613
GetCounterValue		302	304	306	303	305	308
osek_tick_alarm	<default>	133	135	132	133	135	135
	KL	55	52	53	54	59	59
osek_incr_counter		16	16	16	18	18	18
GetActiveApplicationMode		4	4	4	4	3	6
StartOS		2179	2178	2175	2180	2182	2179
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	56	53	56	55	56	56
InitCOM		16	14	14	16	17	18
CloseCOM		16	14	14	16	17	18
StartCOM		50	47	48	140	143	139
StopCOM		34	36	37	34	37	39
ReadFlag		n/a	n/a	n/a	27	24	28
ResetFlag		n/a	n/a	n/a	27	26	28
ReceiveMessage		251	251	253	421	423	425
GetMessageResource		n/a	n/a	n/a	649	650	662
ReleaseMessageResource		n/a	n/a	n/a	626	625	627
GetMessageStatus		n/a	n/a	n/a	198	194	193
SendMessage	SW	831	891	951	999	1045	1113
	NS	869	943	999	1031	1081	1153
	KL	700	750	807	860	915	979

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ActivateTaskset	SW	530	863	979	499	908	922
	NS	554	898	981	536	926	957
	KL	459	782	837	422	795	884
	SW2	530	863	976	499	908	922
	NS2	554	898	981	536	926	957
	KL2	459	784	837	422	795	884
ChainTaskset	SWL	1088	1479	1542	1178	1555	1717
	SWH	1163	1510	1608	1293	1648	1751
	NSL	1104	1485	1585	1220	1616	1733
	NSH	1186	1533	1681	1292	1656	1792
GetTasksetRef		435	448	448	435	437	446
MergeTaskset		173	178	174	176	175	175
AssignTaskset		93	97	104	94	96	95
RemoveTaskset		179	172	174	178	179	181
TestSubTaskset		169	166	168	172	168	171
TestEquivalentTaskset		164	162	165	163	168	166
TickSchedule	SW	300	1027	1078	653	1052	1146
	NS	339	1077	1130	713	1116	1196
	KL	240	955	1009	597	995	1075
	SW2	300	1028	1078	653	1031	1129
	NS2	339	1077	1130	713	1095	1179
	KL2	240	955	1009	597	974	1058
AdvanceSchedule	SW	283	1009	1057	637	1039	1123
	NS	321	1060	1109	691	1095	1179
	KL	214	939	991	572	973	1058
	SW2	283	1009	1057	637	1019	1106
	NS2	321	1060	1109	691	1074	1162
	KL2	214	939	989	572	952	1041
StartSchedule		244	239	240	242	243	242
StopSchedule		170	172	170	171	174	174
GetScheduleStatus		153	156	159	153	154	159
GetScheduleValue		179	186	176	183	182	178
GetScheduleNext		42	41	44	42	43	44
SetScheduleNext		105	105	104	107	109	108
GetArrivalpointDelay		163	159	160	162	163	162
SetArrivalpointDelay		183	184	185	186	185	187

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetArrivalpointTasksetRef		57	57	60	58	59	60
GetArrivalpointNext		59	59	58	60	63	62
SetArrivalpointNext		122	123	125	121	122	126
TestArrivalpointWritable		63	62	63	66	65	65
GetExecutionTime		162	155	157	162	165	161
GetLargestExecutionTime		506	518	519	509	508	521
ResetLargestExecutionTime		493	500	500	493	492	501
GetStackOffset		22	24	22	21	22	22

4.3.2 OS Start-up Time

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

4.3.3 Interrupt Latencies

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	36	36	38	38	38	38
	Cat 2	105	125	129	125	127	129

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	38	36	36	36	38	38
	Cat 2	262	276	271	275	277	284

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	36	36	38	38	36	36
	Cat 2	262	278	276	280	280	278

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.

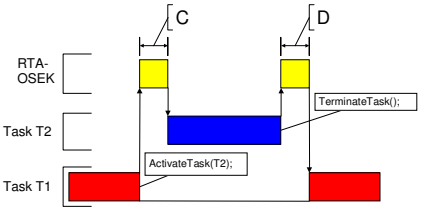

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

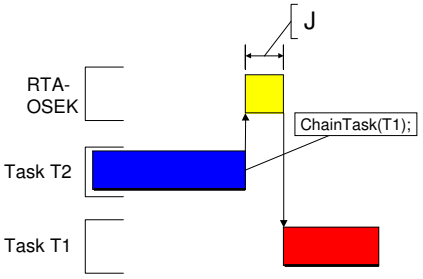

Figure 2: Task Chaining

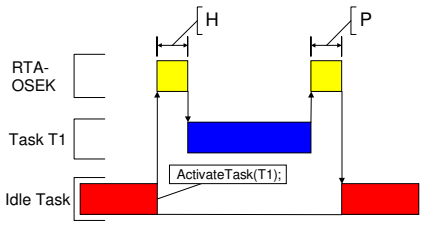

Figure 3: Task Activation from Idle Task

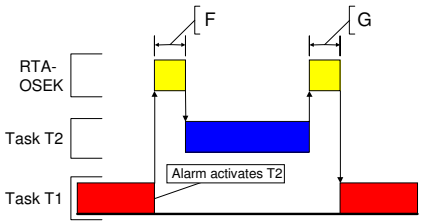

Figure 4: Task Activation from an Alarm

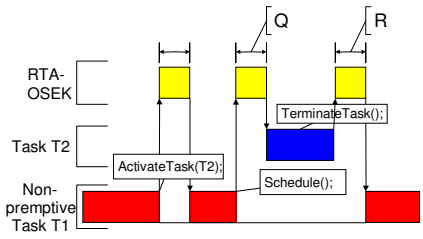

Figure 5: Non-Premptive Task Calls Schedule()

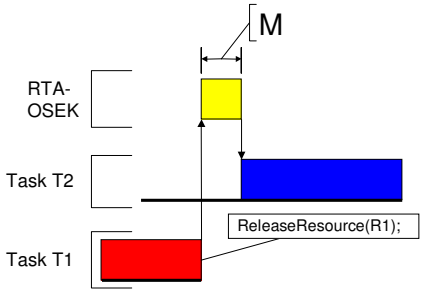

Figure 6: Blocked Task Activated by ReleaseResource()

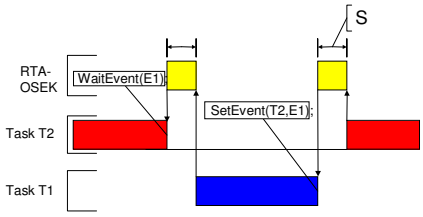

Figure 7: Waiting Task Activated by SetEvent()

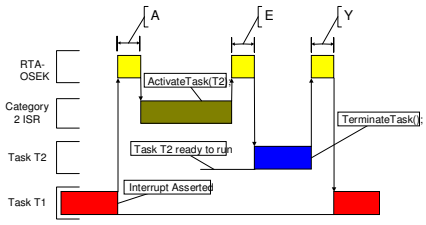

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	110	155	185	111	163	192
Figure 1: D	Heavy, Basic/Extended	178	221	249	237	242	264
ChainTask	Light, Basic	213	299	403	215	302	407
Figure 2: J	Heavy, Basic/Extended	512	635	768	566	651	792
Pre-emption	Light, Basic	163	253	360	167	254	374
Figure 1: C	Heavy, Basic/Extended	243	317	419	358	414	527
From idle task	Light, Basic	164	254	363	169	254	377
Figure 3: H	Heavy, Basic/Extended	244	312	422	360	414	530
Triggered by alarm	Light, Basic	248	338	447	250	338	458
Figure 4: F	Heavy, Basic/Extended	328	396	508	443	496	615
Schedule	Light, Basic	137	171	254	140	169	254
Figure 5: Q	Heavy, Basic/Extended	217	229	313	331	330	413
Release resource	Light, Basic	149	183	237	152	181	240
Figure 6: M	Heavy, Basic/Extended	229	241	296	343	342	399
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	532	529	684
From category 2 ISR	Light, Basic	139	179	237	147	176	238
Figure 8: E	Heavy, Basic/Extended	219	237	296	338	339	397

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	342	390	417	358	398	422
Figure 1: D	Heavy, Basic/Extended	417	447	473	451	446	477
ChainTask	Light, Basic	492	564	664	514	567	672
Figure 2: J	Heavy, Basic/Extended	1012	1099	1233	1078	1113	1261
Pre-emption	Light, Basic	319	381	491	328	385	500
Figure 1: C	Heavy, Basic/Extended	374	440	550	499	550	657
From idle task	Light, Basic	321	382	495	330	386	504

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Figure 3: H	Heavy, Basic/Extended	376	441	554	501	551	661
Triggered by alarm	Light, Basic	403	464	578	412	471	587
Figure 4: F	Heavy, Basic/Extended	460	524	639	586	634	744
Schedule	Light, Basic	292	297	383	300	304	383
Figure 5: Q	Heavy, Basic/Extended	347	356	442	471	469	547
Release resource	Light, Basic	303	309	368	313	314	367
Figure 6: M	Heavy, Basic/Extended	358	368	427	484	479	531
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	652	653	793
From category 2 ISR	Light, Basic	517	539	593	544	543	596
Figure 8: E	Heavy, Basic/Extended	566	592	644	707	702	754

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	481	517	551	481	527	557
Figure 1: D	Heavy, Basic/Extended	547	566	593	574	579	600
ChainTask	Light, Basic	1044	1124	1227	1055	1125	1234
Figure 2: J	Heavy, Basic/Extended	1680	1764	1896	1724	1775	1922
Pre-emption	Light, Basic	742	810	930	752	817	935
Figure 1: C	Heavy, Basic/Extended	803	868	990	922	968	1081
From idle task	Light, Basic	747	812	928	753	819	939
Figure 3: H	Heavy, Basic/Extended	804	868	993	923	972	1085
Triggered by alarm	Light, Basic	879	946	1055	884	952	1069
Figure 4: F	Heavy, Basic/Extended	938	1002	1123	1056	1101	1215
Schedule	Light, Basic	359	356	443	364	373	448
Figure 5: Q	Heavy, Basic/Extended	416	412	508	534	535	608
Release resource	Light, Basic	579	575	636	666	674	731
Figure 6: M	Heavy, Basic/Extended	634	631	701	836	836	891
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1119	1122	1276
From category 2 ISR	Light, Basic	573	578	638	581	592	643
Figure 8: E	Heavy, Basic/Extended	624	632	695	743	746	795

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		68	68	68	68	68	68
BCC1 lightweight, floating-point		76	76	76	76	76	76
BCC1 heavyweight, integer		98	98	98	98	98	98
BCC1 heavyweight, floating-point		98	98	98	98	98	98
BCC2 lightweight, integer		n/a	76	76	n/a	76	76
BCC2 lightweight, floating-point		n/a	76	76	n/a	76	76
BCC2 heavyweight, integer		n/a	98	98	n/a	98	98
BCC2 heavyweight, floating-point		n/a	98	98	n/a	98	98
ECC1 heavyweight, integer		n/a	n/a	n/a	100	100	100
ECC1 heavyweight, floating-point		n/a	n/a	n/a	100	100	100
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	100
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	100
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		70	70	70	70	70	70
BCC1 lightweight, floating-point		78	78	78	78	78	78
BCC1 heavyweight, integer		100	100	100	100	100	100
BCC1 heavyweight, floating-point		100	100	100	100	100	100
BCC2 lightweight, integer		n/a	78	78	n/a	78	78

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
BCC2 lightweight, floating-point	n/a	78	78	n/a	78	78
BCC2 heavyweight, integer	n/a	100	100	n/a	100	100
BCC2 heavyweight, floating-point	n/a	100	100	n/a	100	100
ECC1 heavyweight, integer	n/a	n/a	n/a	102	102	102
ECC1 heavyweight, floating-point	n/a	n/a	n/a	102	102	102
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	102
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	102

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	80	80	80	80	80	80
BCC1 lightweight, floating-point	84	84	84	84	84	84
BCC1 heavyweight, integer	106	106	106	106	106	106
BCC1 heavyweight, floating-point	106	106	106	106	106	106
BCC2 lightweight, integer	n/a	84	84	n/a	84	84
BCC2 lightweight, floating-point	n/a	84	84	n/a	84	84
BCC2 heavyweight, integer	n/a	106	106	n/a	106	106
BCC2 heavyweight, floating-point	n/a	106	106	n/a	106	106
ECC1 heavyweight, integer	n/a	n/a	n/a	110	110	110
ECC1 heavyweight, floating-point	n/a	n/a	n/a	110	110	110
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	110
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	110
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	80	80	82	80	80	82
BCC1 lightweight, floating-point	84	84	86	84	84	86
BCC1 heavyweight, integer	106	106	108	106	106	108
BCC1 heavyweight, floating-point	106	106	108	106	106	108
BCC2 lightweight, integer	n/a	84	86	n/a	84	86

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
BCC2 lightweight, floating-point	n/a	84	86	n/a	84	86
BCC2 heavyweight, integer	n/a	106	108	n/a	106	108
BCC2 heavyweight, floating-point	n/a	106	108	n/a	106	108
ECC1 heavyweight, integer	n/a	n/a	n/a	110	110	112
ECC1 heavyweight, floating-point	n/a	n/a	n/a	110	110	112
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	112
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	112

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	80	80	80	80	80	80
BCC1 lightweight, floating-point	84	84	84	84	84	84
BCC1 heavyweight, integer	106	106	106	106	106	106
BCC1 heavyweight, floating-point	106	106	106	106	106	106
BCC2 lightweight, integer	n/a	84	84	n/a	84	84
BCC2 lightweight, floating-point	n/a	84	84	n/a	84	84
BCC2 heavyweight, integer	n/a	106	106	n/a	106	106
BCC2 heavyweight, floating-point	n/a	106	106	n/a	106	106
ECC1 heavyweight, integer	n/a	n/a	n/a	112	112	112
ECC1 heavyweight, floating-point	n/a	n/a	n/a	112	112	112
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	112
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	112
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	80	80	82	80	80	82
BCC1 lightweight, floating-point	84	84	86	84	84	86
BCC1 heavyweight, integer	106	106	108	106	106	108
BCC1 heavyweight, floating-point	106	106	108	106	106	108
BCC2 lightweight, integer	n/a	84	86	n/a	84	86

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC2 lightweight, floating-point		n/a	84	86	n/a	84	86
BCC2 heavyweight, integer		n/a	106	108	n/a	106	108
BCC2 heavyweight, floating-point		n/a	106	108	n/a	106	108
ECC1 heavyweight, integer		n/a	n/a	n/a	112	112	114
ECC1 heavyweight, floating-point		n/a	n/a	n/a	112	112	114
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	114
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	114

5 CPU Bug CPU_X.003

5.1 CPU_X.003

Upon subroutine calls (i.e. CALLA, CALLR, CALLI, CALLS and PCALL) or interrupt traps the return address and status information is saved on the system stack (e.g. this includes the IP, CSP and PSW). In order to optimize the execution speed of return instructions the C166S V2 core additionally stores (mirrors) the 24 bit return address (i.e. CSP and IP) of the last 6 subroutine calls invoked by CALLA, CALLR, CALLI, CALLS and PCALL in a LIFO (last in first out) return stack. When a RET, RETS or RETP instruction is executed the information from the top of the LIFO is used to optimize fetching the instruction at the return address.

When the actual return address from system stack is available, it is compared with the return stack content and in case of mismatch the instruction obtained from Return stack is discarded and a new fetch from system stack is started.

Problem description: When the return address on system stack is modified and the return stack is not empty a valid instruction must be located at:

1) Address CSP, IP located on the return stack top when the next return instruction is RETS.

2) Address on current CSP and IP located on the return stack top when the next instruction is RET or RETP.

Affected Devices: All devices and steps of the XC16x family are affected by this. The XC2xxx devices are not affected.

5.2 Workaround 3 (Avoid Generating PACER trap)

Use a RETI instruction to branch to modified return address after manipulating system stack. The RETI instruction does not affect the return stack. In this case, the PSW must be pushed on to the system stack before pushing modified return address on to the return registers.

```
PUSH PSW          ; push current contents of PSW
PUSH Rx           ; push return segment address CSP
PUSH Ry           ; push return address IP
RETI              ; branch (=return) to CSP:IP
```

5.3 Fix for CPU_X.003

The RTA-OSEK provides a fix for CPU Bug x.003 using workaround 3. To activate this the command line option -DOS_RETS_FIX should be used when assembling osgen.src. The example application demonstrates this. To enable the fix open example.oil from the RTA-OSEK GUI and go to Builder, Custom Build, click configure and select Build Script uncomment the following line by removing the rem.

```
rem %as% -DOS_RETS_FIX -Wa--no-warnings=771 -co -
C%CPU_TYPE% --list-file -t osgen.src
```

5.4 Changes from v5.0.0

The v5.0.1 release has the following changes from the v5.0.0 release:

The kernel is now built and tested with Tasking VX Toolchain V2.3r2 instead of v2.2r1.

ORTI debugging is now supported for Lauterbach TRACE32

Data initialized by compiler Startup code and StartOS() API is now contained in different sections. The new sections os_data2 has been created to hold variables that are initialized by the C startup. The existing sections os_pur and os_pir now only hold variables initialized by the StartOS API call.

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.