RTA-OSEK

Binding Manual: S12X/COSMIC

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: M00078-002

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

2.1.1 Background

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

Placement of code and data into memory pages is controlled by the linker file. See the Cosmic Compiler Manual for details.

In addition to its 16-bit logical address space the S12X also has a 23-bit global address space. This global address space contains all physical memory. That is, all of the Flash, RAM and EEPROM pages as well as the unbanked memory are simultaneously visible in the global address space.

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

2.1.2 Function Addresses

In the banked memory model functions have 4-byte addresses, with the upper word contains 2-byte logical address. The MSB of lower word contains `PPAGE` number and LSB of lower word is always 0. The PPAGE number indicates the page of Flash that should be mapped into the banked memory-

window between 0x8000 and 0xBFFF. The currently-mapped page is stored in the PPAGE register and the assembly instructions CALL and RTC maintain this implicitly.

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

2.1.3 Data Addresses

In the banked memory model, data is "Near" by default. "Near" data only has a 2-byte address and consequently pointers to near data are 2 bytes. Near data must be placed in an unbanked memory area.

2.1.4 Preservation of PAGE Registers by ISRs

In the banked memory model, it is not necessary to preserve the PPAGE, EPAGE, GPAGE and RPAGE registers during an ISR. If the application code either directly or indirectly uses the page registers then Category 2 ISR's should use the floating point wrappers to correctly handle these registers.

2.1.5 Additional Preservation of PAGE Registers

Modification of the EPAGE, GPAGE and RPAGE registers within an application is supported by RTA-OSEK with the floating point wrappers. These wrappers are found in the files osfptgt.c and osfptgt.h in the <RTA-OSEK install dir>\COS12X\inc folder. The default implementation supplied with the port demonstrates how the GPAGE, EPAGE and RPAGE registers can be preserved. To apply this additional context saving by the wrappers all tasks and ISRs that modify the PAGE registers should be marked as using floating point in the RTA-OSEK GUI. Further details on the floating point wrappers can be found in the RTA-OSEK User Guide.

2.2 Compiler

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

Vendor	Cosmic
Compiler	cxs12x
Version	V4.7.10

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

Option	Description
+modf	Compile all functions as far
+nowiden	Do not widen char parameters to integers

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
+nowiden	Do not widen char parameters to integers
+debug	Generate debug information
+modf	Compile all functions as far

2.3 Assembler

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

Vendor	Cosmic
Assembler	cas12x
Version	V4.5.2

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.

2.4 Linker/Locator

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

Option	Description
+def os_ppage=<value>	<value>=the address of PPAGE if any, or otherwise any unused RAM address

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_intvec	ROM	Vector table (if generated)
.os_intvec1	ROM	Non-relocatable Vector

Sections	ROM/RAM	Description
		table (if generated)
.os_pir	RAM	RTA-OSEK initialized data; initialized during StartOS()
.os_pir2	RAM	RTA-OSEK initialized data; must be initialized during C-startup
.os_pur	RAM	RTA-OSEK uninitialized data; zeroed during StartOS()
.os_pur2	RAM	RTA-OSEK uninitialized data; must be zeroed during C-startup
.os_ftext	Banked text ROM	RTA-OSEK code
.os_unbank	unbanked text ROM	RTA-OSEK interrupt handling code
.os_trace_ram	RAM	RTA-TRACE Buffer

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

C Library Functions	Description
crtsx.s	startup code with automatic initilization
c_lgsub	evaluates the (long) difference between the value pointed at by the Y register and the value in long register

This port of RTA-OSEK is built for the libm.x12,libi.x12 runtime libraries. The compiler flag +modf specifies all functions to be @far (banked memory model).

2.4.1 Compiler Sections

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

RTA-OSEK Section	Section Contents
.os_ftext	Code – located in banked memory.
.os_unbank	Interrupt handling code – must be located in unbanked memory.
.os_pid	Read-only data – must be located in unbanked memory.
.os_pird	Read-only data – must be located in unbanked memory.
.os_const	Read-only data – must be located in

	unbanked memory.
.os_pur	Writeable data – must be located in unbanked RAM.
.os_pur2	Writeable data – must be located in unbanked RAM.
.os_pir	Writeable data – must be located in unbanked RAM.
.os_pir2	Writeable data – must be located in unbanked RAM.
.os_trace_ram	Trace Buffer – located in banked RAM.
.os_intvec	Vector table – must be located in unbanked memory.
.os_intvec1	High vector table – must be located in unbanked memory.

2.4.2 Section Placement

The vector table must be located in **unbanked** memory. If the vector table is generated by RTA-OSEK then it is in sections `os_intvec` and `os_intvec1`, which must be located in unbanked memory with a `+SEG` clause of the linker file.

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

All other RTA-OSEK code is in the `os_ftext` section. This section should be placed in banked memory.

Read-only variables are in the `.os_pid`, `.os_pird` and `.os_const` sections, which should appear in a `+SEG` section of the linker file and be mapped to unbanked memory. E.g. the unbanked flash areas `0x4000..0x7FFF` or `0xC000..0xFF0F`.

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

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
HiWare HiWave
iSYSTEM winIDEA

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for S12X/COSMIC. 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

S12XCPUV1 Reference Manual.

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

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

3.1.2 Interrupt Vectors

For the allocation of Category 1 and Category 2 interrupt handlers to interrupt vectors on your target hardware, the following restrictions apply:

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

The valid base addresses for the vector table are:

Base Register	Notes
0xFF	Default base address
0x40..0x7F, 0xC0..0xFE	Unbanked Flash
0x80..0xBF	Banked Flash. Only valid if application does not use multiple flash pages.
0x08..0x0B, 0x10..0x3F	Banked EEPROM; paged and unpaged SRAM. Not recommended. User is responsible for setup and for managing side-effects.
0x0C..0x0F	Unbanked EEPROM. Not recommended.

3.1.3 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Cosmic S12X 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 (VV represents the 2 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
0xVV	_os_wrapper_VV
e.g. 0x90	_os_wrapper_90

The S12X has some vectors that have a fixed location (vectors 0xFFFC, 0xFFFA) and the rest of the vectors are relocatable. If a vector table is generated it is in two sections. Section os_intvec1 contains the fixed location vectors and must be located at 0xFFFA. Section os_intvec contains the relocatable vectors and can be placed at any of the locations outlined in the processor documentation. The user is responsible for initializing the Interrupt Vector Base Register (IVBR).

3.1.6 Interrupt Priority Levels

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

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

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

The init_target() function in target.c in the example application, located in <RTA-OSEK install directory>\COS12X\Example\, gives an example of how to copy os_InitIrqLevels to the correct location.

3.2 Register Settings

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

Register	Value	Notes
CCRH	os_oim	Set the IPL to block out all category 2 interrupts
CCR.I	0	clear the I bit in CCR
CCR.U	0	clear the U bit in CCR to operate in the supervisor mode
INT_CFDATA*x*	0	Initialize interrupt priority table

The RTA-OSEK Component does not reserve the use of any hardware registers.

3.3 Stack Usage

3.3.1 Number of Stacks

A single stack is used. The first argument to `StackFaultHook` is always 0.

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

3.3.2 Stack Usage within API Calls

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

Standard

API max usage (bytes): 32

Timing

API max usage (bytes): 32

Extended

API max usage (bytes): 32

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 Initialization of the stack pointer

The S12X instructions that push data onto the stack decrement the `SP` register before using it. Therefore the start-up code provided with the compiler initializes the `SP` to the value of `__stack`, which is a linker-defined constant equal to the address of the first byte beyond the stack area.

For example, if the linker command file specifies that the `STACK` section should be placed in memory at addresses `0x3000-0x3FFF`, then the `SP` will be initialized to `0x4000`. A subsequent stack 'push' will therefore write to memory at `0x3FFF` (and not `0x4000`).

The user has to define `__stack` or a similar symbol in the linker file which is used by the start-up code (crtsx.s) to initialize Stack pointer to the address specified by symbol `__stack`.

3.4 User vs. Supervisor State

The latest revision of the CPU ('S12XCPUV2'), as found in the S12XE family, features the ability to run in a protected 'user state' (as opposed to supervisor state) by setting the new U bit in the CCR.

Important: RTA-OSEK requires the CPU to operate in the supervisor state at all times. Applications must therefore take care never to set the U bit of the CCR.

Register	Value	Notes
`CCR.U`	0	CPU in Supervisor state
`CCR.U`	1	CPU in user state

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	**MC9S12XDP512**
Clock speed (MHz)	16
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	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	112	112	117	197	197	202
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	182	262	262	267
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	51	51	51	51	51	51
	ROM	202	202	207	287	287	292
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	23	23	23	23	23	23
BCC1 Heavyweight task	RAM	2	2	2	2	2	2
	ROM	25	25	25	25	25	25
BCC2 task	RAM	n/a	3	5	n/a	3	5
	ROM	n/a	28	32	n/a	28	32
ECC1, Integer task	RAM	n/a	n/a	n/a	11	11	11
	ROM	n/a	n/a	n/a	35	35	35
ECC1, floating-point task	RAM	n/a	n/a	n/a	14	14	14
	ROM	n/a	n/a	n/a	35	35	35
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	13
	ROM	n/a	n/a	n/a	n/a	n/a	39
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	16
	ROM	n/a	n/a	n/a	n/a	n/a	39
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	31	31	31	31	31	31
Category 2 ISR, floating-point	RAM	3	3	3	3	3	3
	ROM	43	43	43	43	43	43
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	9	9	9	9	9	9
	ROM	31	31	31	31	31	31
Counter	RAM	4	4	4	4	4	4
	ROM	82	82	82	82	82	82
Message	RAM	11	11	11	31	31	31
	ROM	12	12	12	29	29	29
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
ScheduleTable	RAM	9	9	9	9	9	9
	ROM	86	86	86	86	86	86
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	11	11	11	11	11	11
	ROM	24	24	24	24	24	24
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

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	9	9	9	9	9	9
	ROM	30	30	30	30	30	30
BCC1 Heavyweight task	RAM	11	11	11	11	11	11
	ROM	32	32	32	32	32	32
BCC2 task	RAM	n/a	12	14	n/a	12	14
	ROM	n/a	35	39	n/a	35	39
ECC1, Integer task	RAM	n/a	n/a	n/a	20	20	20
	ROM	n/a	n/a	n/a	42	42	42
ECC1, floating-point task	RAM	n/a	n/a	n/a	23	23	23
	ROM	n/a	n/a	n/a	42	42	42
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	22

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	46
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	25
	ROM	n/a	n/a	n/a	n/a	n/a	46
Category 2 ISR	RAM	9	9	9	9	9	9
	ROM	53	53	53	53	53	53
Category 2 ISR, floating-point	RAM	12	12	12	12	12	12
	ROM	61	61	61	61	61	61
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	9	9	9	9	9	9
	ROM	31	31	31	31	31	31
Counter	RAM	4	4	4	4	4	4
	ROM	82	82	82	82	82	82
Message	RAM	11	11	11	31	31	31
	ROM	12	12	12	29	29	29
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
ScheduleTable	RAM	9	9	9	9	9	9
	ROM	86	86	86	86	86	86
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	11	11	11	11	11	11

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	24	24	24	24	24	24
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

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	10	10	10	10	10	10
	ROM	34	34	34	34	34	34
BCC1 Heavyweight task	RAM	12	12	12	12	12	12
	ROM	34	34	34	34	34	34
BCC2 task	RAM	n/a	13	15	n/a	13	15
	ROM	n/a	37	41	n/a	37	41
ECC1, Integer task	RAM	n/a	n/a	n/a	21	21	21
	ROM	n/a	n/a	n/a	44	44	44
ECC1, floating-point task	RAM	n/a	n/a	n/a	24	24	24
	ROM	n/a	n/a	n/a	44	44	44
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	23
	ROM	n/a	n/a	n/a	n/a	n/a	48
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	26
	ROM	n/a	n/a	n/a	n/a	n/a	48
Category 2 ISR	RAM	10	10	10	10	10	10
	ROM	57	57	57	57	57	57
Category 2 ISR, floating-point	RAM	13	13	13	13	13	13
	ROM	65	65	65	65	65	65
Resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Alarm	RAM	9	9	9	9	9	9
	ROM	33	33	33	33	33	33
Counter	RAM	4	4	4	4	4	4
	ROM	84	84	84	84	84	84
Message	RAM	11	11	11	31	31	31
	ROM	14	14	14	31	31	31
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
ScheduleTable	RAM	9	9	9	9	9	9
	ROM	86	86	86	86	86	86
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	15	15	15	15	15	15
	ROM	30	30	30	30	30	30
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	98	143	185	102	151	203
	NS		85	130	172	89	138	190
	KL	2	57	102	146	61	110	164
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	17	17	17	17	17	17
ChainTask	SWL	1, 8	84	129	173	88	137	191
	SWH	1, 9	112	155	199	116	163	217
	NSL	8	84	129	173	88	137	191
	NSH	9	106	149	193	110	157	211
Schedule			58	58	77	58	58	77
GetTaskID			23	23	23	23	23	23
GetTaskState			60	60	60	72	72	72
EnableAllInterrupts			8	8	8	8	8	8
DisableAllInterrupts			21	21	21	21	21	21
ResumeAllInterrupts			13	13	13	13	13	13
SuspendAllInterrupts			29	29	29	29	29	29
ResumeOSInterrupts			13	13	13	13	13	13
SuspendOSInterrupts			36	36	36	36	36	36
GetResource	Task	7	27	27	30	27	27	30
	Combined	6	67	67	67	67	67	67
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	49	49	49	49	49	49
	Combined	6	92	92	92	92	92	92
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	91	91	168
	NS		n/a	n/a	n/a	78	78	155
	NS1i	10	n/a	n/a	n/a	37	n/a	n/a
	KL	2	n/a	n/a	n/a	57	57	135
	KL1i	2, 10	n/a	n/a	n/a	14	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	181	181	352
	fp	11	n/a	n/a	n/a	205	205	406
	1i	10	n/a	n/a	n/a	15	n/a	n/a
GetAlarmBase			42	42	42	42	42	42
GetAlarm			89	89	89	89	89	89
SetRelAlarm			706	706	706	706	706	706
SetAbsAlarm			922	922	922	922	922	922
CancelAlarm			60	60	60	60	60	60
InitCounter			61	61	61	61	61	61
GetCounterValue			77	77	77	77	77	77
GetScheduleTableStatus		34	63	73	73	63	73	73
NextScheduleTable		34	74	151	151	74	151	151
StartScheduleTable		34	115	171	171	115	171	171
StopScheduleTable		34	69	84	84	69	84	84
ScheduleTable expiry point	ActivateTask		30	30	30	30	30	30
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		5	5	5	5	5	5
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		26	26	26	26	26	26
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	25	25	25	25	25	25
Process container	Non-Yielding	33	13	13	13	13	13	13
osek_tick_alarm	<default>		73	73	73	73	73	73
	KL	2	51	51	51	51	51	51
osek_incr_counter			56	56	56	56	56	56
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			100	100	100	100	100	100
ShutdownOS	NoHook	12	18	18	18	18	18	18
	Hook	13	25	25	25	25	25	25
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			18	18	18	18	18	18
StopCOM			18	18	18	18	18	18
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	50	50	50	162	162	162

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	162	162	162	162	162	162
GetMessageResource			23	23	23	23	23	23
ReleaseMessageResource			21	21	21	21	21	21
GetMessageStatus			48	48	48	48	48	48
SendMessage	SW CCCA	1, 14	74	74	74	204	204	204
	SW CCCB	1, 15	182	182	182	204	204	204
	NS CCCA	14	74	74	74	204	204	204
	NS CCCB	15	182	182	182	204	204	204
	KL CCCA	2, 14	57	57	57	191	191	191
	KL CCCB	2, 15	169	169	169	191	191	191
main_dispatch	NoHook	12	126	126	155	126	126	155
	Hook	13	153	153	184	153	153	184
sub_dispatch	B1LF	19	16	16	16	16	16	16
	B1HI	20	57	57	57	57	57	57
	B1HF	21	65	65	65	65	65	65
	B2LI	22	n/a	39	66	n/a	39	66
	B2LF	23	n/a	47	74	n/a	47	74
	B2HI	24	n/a	157	227	n/a	157	227
	B2HF	25	n/a	165	235	n/a	165	235
	E1HI	26	n/a	n/a	n/a	249	249	330
	E1HF	27	n/a	n/a	n/a	257	257	338
	E2HI	28	n/a	n/a	n/a	n/a	n/a	330
	E2HF	29	n/a	n/a	n/a	n/a	n/a	338
ErrorHook support		16	20	20	20	20	20	20
	ServiceID	17	25	25	25	25	25	25
	Parameters	18	58	58	58	58	58	58
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	96	200	248	106	225	290
	NS		83	187	235	93	212	277
	KL	2	55	161	209	65	186	251
ChainTaskset	SWL	1, 8	84	202	249	95	226	291
	SWH	1, 9	122	248	295	131	272	337
	NSL	8	84	202	249	95	226	291
	NSH	9	116	242	289	125	266	331
GetTasksetRef			10	10	10	10	10	10

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
MergeTaskset			36	36	36	36	36	36
AssignTaskset			10	10	10	10	10	10
RemoveTaskset			38	38	38	38	38	38
TestSubTaskset			47	47	47	47	47	47
TestEquivalentTaskset			44	44	44	44	44	44
TickSchedule	SW	1	181	156	156	156	156	156
	NS		163	140	140	140	140	140
	KL	2	148	125	125	125	125	125
AdvanceSchedule	SW	1	158	131	131	131	131	131
	NS		142	115	115	115	115	115
	KL	2	123	94	94	94	94	94
StartSchedule			64	64	64	64	64	64
StopSchedule			40	40	40	40	40	40
GetScheduleStatus			77	77	77	77	77	77
GetScheduleValue			65	65	65	65	65	65
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			12	12	12	12	12	12
GetArrivalpointDelay			37	37	37	37	37	37
SetArrivalpointDelay			35	35	35	35	35	35
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			10	10	10	10	10	10
TestArrivalpointWritable			21	21	21	21	21	21
GetExecutionTime			5	5	5	5	5	5
GetLargestExecutionTime			9	9	9	9	9	9
ResetLargestExecutionTime			2	2	2	2	2	2
GetStackOffset			21	21	21	21	21	21

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	98	143	185	102	151	203
	NS		85	130	172	89	138	190
	KL	2	57	102	146	61	110	164
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	17	17	17	17	17	17
ChainTask	SWL	1, 8	84	129	173	88	137	191
	SWH	1, 9	112	155	199	116	163	217
	NSL	8	84	129	173	88	137	191
	NSH	9	106	149	193	110	157	211
Schedule			77	77	96	77	77	96
GetTaskID			23	23	23	23	23	23
GetTaskState			60	60	60	72	72	72
EnableAllInterrupts			8	8	8	8	8	8
DisableAllInterrupts			21	21	21	21	21	21
ResumeAllInterrupts			13	13	13	13	13	13
SuspendAllInterrupts			29	29	29	29	29	29
ResumeOSInterrupts			13	13	13	13	13	13
SuspendOSInterrupts			36	36	36	36	36	36
GetResource	Task	7	27	27	30	27	27	30
	Combined	6	67	67	67	67	67	67
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	68	68	68	68	68	68
	Combined	6	130	130	130	130	130	130
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	91	91	168
	NS		n/a	n/a	n/a	78	78	155
	NS1i	10	n/a	n/a	n/a	37	n/a	n/a
	KL	2	n/a	n/a	n/a	57	57	135
	KL1i	2, 10	n/a	n/a	n/a	14	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	224	224	392
	fp	11	n/a	n/a	n/a	248	248	446

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	54	n/a	n/a
GetAlarmBase			42	42	42	42	42	42
GetAlarm			89	89	89	89	89	89
SetRelAlarm			706	706	706	706	706	706
SetAbsAlarm			922	922	922	922	922	922
CancelAlarm			60	60	60	60	60	60
InitCounter			61	61	61	61	61	61
GetCounterValue			77	77	77	77	77	77
GetScheduleTableStatus		34	63	73	73	63	73	73
NextScheduleTable		34	74	151	151	74	151	151
StartScheduleTable		34	115	171	171	115	171	171
StopScheduleTable		34	69	84	84	69	84	84
ScheduleTable expiry point	ActivateTask		30	30	30	30	30	30
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		5	5	5	5	5	5
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		26	26	26	26	26	26
GetISRID		4	29	29	29	29	29	29
Process container	Yielding	32	25	25	25	25	25	25
Process container	Non-Yielding	33	13	13	13	13	13	13
osek_tick_alarm	<default>		73	73	73	73	73	73
	KL	2	51	51	51	51	51	51
osek_incr_counter			56	56	56	56	56	56
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			136	136	136	136	136	136
ShutdownOS	NoHook	12	18	18	18	18	18	18
	Hook	13	25	25	25	25	25	25
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			18	18	18	18	18	18
StopCOM			18	18	18	18	18	18
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	50	50	50	162	162	162
	CCCB	15	162	162	162	162	162	162
GetMessageResource			23	23	23	23	23	23
ReleaseMessageResource			21	21	21	21	21	21

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			48	48	48	48	48	48
SendMessage	SW CCCA	1, 14	74	74	74	204	204	204
	SW CCCB	1, 15	182	182	182	204	204	204
	NS CCCA	14	74	74	74	204	204	204
	NS CCCB	15	182	182	182	204	204	204
	KL CCCA	2, 14	57	57	57	191	191	191
	KL CCCB	2, 15	169	169	169	191	191	191
main_dispatch	NoHook	12	182	182	210	182	182	210
	Hook	13	211	211	242	211	211	242
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	63	63	63	63	63	63
	B1HF	21	71	71	71	71	71	71
	B2LI	22	n/a	29	56	n/a	29	56
	B2LF	23	n/a	37	64	n/a	37	64
	B2HI	24	n/a	149	219	n/a	149	219
	B2HF	25	n/a	157	227	n/a	157	227
	E1HI	26	n/a	n/a	n/a	274	274	353
	E1HF	27	n/a	n/a	n/a	282	282	361
	E2HI	28	n/a	n/a	n/a	n/a	n/a	353
	E2HF	29	n/a	n/a	n/a	n/a	n/a	361
ErrorHook support		16	20	20	20	20	20	20
	ServiceID	17	25	25	25	25	25	25
	Parameters	18	58	58	58	58	58	58
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	45	45	45	45	45	45
Timing_termination		4	66	66	66	66	66	66
ActivateTaskset	SW	1	96	200	248	106	225	290
	NS		83	187	235	93	212	277
	KL	2	55	161	209	65	186	251
ChainTaskset	SWL	1, 8	84	202	249	95	226	291
	SWH	1, 9	122	248	295	131	272	337
	NSL	8	84	202	249	95	226	291
	NSH	9	116	242	289	125	266	331
GetTasksetRef			10	10	10	10	10	10
MergeTaskset			36	36	36	36	36	36
AssignTaskset			10	10	10	10	10	10
RemoveTaskset			38	38	38	38	38	38

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			47	47	47	47	47	47
TestEquivalentTaskset			44	44	44	44	44	44
TickSchedule	SW	1	181	156	156	156	156	156
	NS		163	140	140	140	140	140
	KL	2	148	125	125	125	125	125
AdvanceSchedule	SW	1	158	131	131	131	131	131
	NS		142	115	115	115	115	115
	KL	2	123	94	94	94	94	94
StartSchedule			64	64	64	64	64	64
StopSchedule			40	40	40	40	40	40
GetScheduleStatus			77	77	77	77	77	77
GetScheduleValue			65	65	65	65	65	65
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			12	12	12	12	12	12
GetArrivalpointDelay			37	37	37	37	37	37
SetArrivalpointDelay			35	35	35	35	35	35
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			10	10	10	10	10	10
SetArrivalpointNext			10	10	10	10	10	10
TestArrivalpointWritable			21	21	21	21	21	21
GetExecutionTime			79	79	79	79	79	79
GetLargestExecutionTime			41	41	41	41	41	41
ResetLargestExecutionTime			37	37	37	37	37	37
GetStackOffset			21	21	21	21	21	21

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	171	220	262	173	224	280
	NS		210	258	300	214	266	319
	KL	2	125	174	216	127	178	233
TerminateTask	LExt	3	72	72	72	72	72	72

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	95	95	95	95	95	95
ChainTask	SWL	1, 8	196	245	288	200	253	306
	SWH	1, 9	226	273	317	230	281	335
	NSL	8	251	306	349	255	314	367
	NSH	9	275	326	370	279	336	391
Schedule			155	155	175	155	155	175
GetTaskID			33	33	33	33	33	33
GetTaskState			169	169	169	171	171	171
EnableAllInterrupts			18	18	18	18	18	18
DisableAllInterrupts			31	31	31	31	31	31
ResumeAllInterrupts			46	46	46	46	46	46
SuspendAllInterrupts			39	39	39	39	39	39
ResumeOSInterrupts			49	49	49	49	49	49
SuspendOSInterrupts			46	46	46	46	46	46
GetResource	Task	7	271	271	240	271	271	240
	Combined	6	237	237	237	237	237	237
	CLEx	3	235	235	235	235	235	235
ReleaseResource	Task	7	251	251	251	251	251	251
	Combined	6	302	302	302	302	302	302
	CLEx	3	220	220	220	220	220	220
SetEvent	SW	1	n/a	n/a	n/a	211	211	290
	NS		n/a	n/a	n/a	251	251	330
	NS1i	10	n/a	n/a	n/a	156	n/a	n/a
	KL	2	n/a	n/a	n/a	165	165	244
	KL1i	2, 10	n/a	n/a	n/a	121	n/a	n/a
ClearEvent			n/a	n/a	n/a	85	85	85
GetEvent			n/a	n/a	n/a	121	121	121
WaitEvent	<default>		n/a	n/a	n/a	298	298	462
	fp	11	n/a	n/a	n/a	322	322	516
	1i	10	n/a	n/a	n/a	136	n/a	n/a
GetAlarmBase			125	125	125	125	125	125
GetAlarm			138	138	138	138	138	138
SetRelAlarm			838	838	838	838	838	838
SetAbsAlarm			1034	1034	1034	1034	1034	1034
CancelAlarm			110	110	110	110	110	110
InitCounter			187	187	187	187	187	187
GetCounterValue			160	160	160	160	160	160

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	73	83	83	73	83	83
NextScheduleTable		34	84	161	161	84	161	161
StartScheduleTable		34	125	181	181	125	181	181
StopScheduleTable		34	79	94	94	79	94	94
ScheduleTable expiry point	ActivateTask		30	30	30	30	30	30
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		5	5	5	5	5	5
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		26	26	26	26	26	26
GetISRID		4	38	38	38	38	38	38
Process container	Yielding	32	25	25	25	25	25	25
Process container	Non-Yielding	33	13	13	13	13	13	13
osek_tick_alarm	<default>		92	92	92	92	92	92
	KL	2	51	51	51	51	51	51
osek_incr_counter			56	56	56	56	56	56
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			146	146	146	146	146	146
ShutdownOS	NoHook	12	23	23	23	23	23	23
	Hook	13	30	30	30	30	30	30
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			28	28	28	28	28	28
StopCOM			33	33	33	33	33	33
ReadFlag			26	26	26	26	26	26
ResetFlag			23	23	23	23	23	23
ReceiveMessage	CCCA	14	109	109	109	219	219	219
	CCCB	15	219	219	219	219	219	219
GetMessageResource			64	64	64	64	64	64
ReleaseMessageResource			64	64	64	64	64	64
GetMessageStatus			72	72	72	72	72	72
SendMessage	SW CCCA	1, 14	139	139	139	271	271	271
	SW CCCB	1, 15	249	249	249	271	271	271
	NS CCCA	14	139	139	139	271	271	271
	NS CCCB	15	249	249	249	271	271	271
	KL CCCA	2, 14	108	108	108	240	240	240
	KL CCCB	2, 15	218	218	218	240	240	240
main_dispatch	NoHook	12	182	182	210	182	182	210

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	211	211	242	211	211	242
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	63	63	63	63	63	63
	B1HF	21	71	71	71	71	71	71
	B2LI	22	n/a	29	57	n/a	29	57
	B2LF	23	n/a	37	65	n/a	37	65
	B2HI	24	n/a	149	221	n/a	149	221
	B2HF	25	n/a	157	229	n/a	157	229
	E1HI	26	n/a	n/a	n/a	274	274	355
	E1HF	27	n/a	n/a	n/a	282	282	363
	E2HI	28	n/a	n/a	n/a	n/a	n/a	355
	E2HF	29	n/a	n/a	n/a	n/a	n/a	363
ErrorHook support		16	72	72	72	72	72	72
	ServiceID	17	78	78	78	78	78	78
	Parameters	18	108	108	108	108	108	108
validity_checks		3	30	30	30	30	30	30
Timing_dispatch		4	45	45	45	45	45	45
Timing_termination		4	66	66	66	66	66	66
ActivateTaskset	SW	1	219	263	311	227	287	343
	NS		255	299	347	263	323	379
	KL	2	171	216	265	180	241	297
ChainTaskset	SWL	1, 8	267	315	363	275	337	393
	SWH	1, 9	300	359	407	308	381	437
	NSL	8	312	360	408	320	382	438
	NSH	9	339	398	446	347	420	476
GetTasksetRef			100	100	100	100	100	100
MergeTaskset			168	168	168	168	168	168
AssignTaskset			121	121	121	121	121	121
RemoveTaskset			170	170	170	170	170	170
TestSubTaskset			175	175	175	175	175	175
TestEquivalentTaskset			172	172	172	172	172	172
TickSchedule	SW	1	276	238	238	238	238	238
	NS		320	303	303	303	303	303
	KL	2	233	195	195	195	195	195
AdvanceSchedule	SW	1	275	233	233	233	233	233
	NS		319	302	302	302	302	302
	KL	2	234	188	188	188	188	188

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			172	172	172	172	172	172
StopSchedule			118	118	118	118	118	118
GetScheduleStatus			156	156	156	156	156	156
GetScheduleValue			160	160	160	160	160	160
GetScheduleNext			66	66	66	66	66	66
SetScheduleNext			117	117	117	117	117	117
GetArrivalpointDelay			131	131	131	131	131	131
SetArrivalpointDelay			150	150	150	150	150	150
GetArrivalpointTasksetRef			85	85	85	85	85	85
GetArrivalpointNext			89	89	89	89	89	89
SetArrivalpointNext			127	127	127	127	127	127
TestArrivalpointWritable			100	100	100	100	100	100
GetExecutionTime			108	108	108	108	108	108
GetLargestExecutionTime			131	131	131	131	131	131
ResetLargestExecutionTime			117	117	117	117	117	117
GetStackOffset			21	21	21	21	21	21

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 S12X/COSMIC port of the RTA-OSEK Component was achieved using a timer running 2 times slower than the CPU clock speed. The figures in this section, therefore, have an uncertainty level of up to 2 CPU cycles. The actual times are between 0 and 2 cycles shorter than those reported in the remainder of this section.

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	169	269	357	181	251	387
	NS	155	255	343	167	237	373
	KL	125	221	311	137	207	345
TerminateTask	LExt	0	0	0	0	0	0
	H	241	245	259	245	245	259
ChainTask	SWL	433	559	709	535	591	817
	SWH	535	657	805	639	689	925
	NSL	433	571	709	535	603	817
	NSH	525	647	795	629	679	903
Schedule	SW	141	145	167	145	145	167
GetTaskID		93	95	95	95	95	95
GetTaskState		157	159	155	181	179	177
EnableAllInterrupts		39	39	39	39	39	39
DisableAllInterrupts		73	61	61	73	61	61
ResumeAllInterrupts		49	49	49	49	49	49
SuspendAllInterrupts		77	77	77	77	77	77
ResumeOSInterrupts		49	49	49	49	49	49
SuspendOSInterrupts		77	77	77	77	77	77
GetResource	Task	97	109	113	103	115	119
	Combined	115	115	115	121	121	121
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	125	125	125	129	129	129
	Combined	143	143	143	149	149	149
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	215	203	201
	NS	n/a	n/a	n/a	203	203	201
	KL	n/a	n/a	n/a	161	161	161
ClearEvent		n/a	n/a	n/a	99	99	99
GetEvent		n/a	n/a	n/a	73	73	73
WaitEvent	<default>	n/a	n/a	n/a	681	659	803

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	fp	n/a	n/a	n/a	701	677	823
GetAlarmBase		225	227	227	213	213	213
GetAlarm		189	195	195	189	189	189
SetRelAlarm		289	299	299	289	289	289
SetAbsAlarm		259	269	269	259	259	259
CancelAlarm		121	123	123	121	121	121
InitCounter		161	167	167	161	161	161
GetCounterValue		161	167	167	161	161	161
osek_tick_alarm	<default>	223	225	225	223	223	223
	KL	167	169	169	167	167	167
osek_incr_counter		37	39	39	37	37	37
GetActiveApplicationMode		25	25	13	13	13	13
StartOS		1255	1257	1257	1257	1257	1257
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	65	65	65	65	65	65
InitCOM		43	43	43	43	43	43
CloseCOM		31	31	31	31	31	31
StartCOM		105	105	105	429	429	429
StopCOM		65	65	65	65	65	65
ReadFlag		n/a	n/a	n/a	21	21	21
ResetFlag		n/a	n/a	n/a	15	15	15
ReceiveMessage		119	123	123	535	535	535
GetMessageResource		n/a	n/a	n/a	213	213	213
ReleaseMessageResource		n/a	n/a	n/a	221	221	221
GetMessageStatus		n/a	n/a	n/a	127	127	127
SendMessage	SW	321	431	519	725	795	931
	NS	301	411	499	717	787	923
	KL	237	343	433	647	729	855
ActivateTaskset	SW	141	739	969	141	733	1007
	NS	115	713	943	127	719	993
	KL	85	683	913	97	689	963
	SW2	129	727	957	141	733	1007
	NS2	115	713	943	127	719	993
	KL2	85	683	913	97	689	963
ChainTaskset	SWL	409	1035	1325	501	1081	1443
	SWH	515	1143	1433	607	1189	1551

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NSL	409	1035	1325	501	1081	1443
	NSH	505	1133	1423	597	1179	1541
GetTasksetRef		63	65	63	65	63	65
MergeTaskset		115	115	115	115	115	115
AssignTaskset		63	63	63	63	63	63
RemoveTaskset		119	119	119	119	119	119
TestSubTaskset		139	139	139	139	139	139
TestEquivalentTaskset		131	131	131	143	131	131
TickSchedule	SW	363	1053	1297	455	1087	1321
	NS	333	1015	1259	429	1049	1295
	KL	303	985	1229	399	1019	1265
	SW2	363	1043	1273	455	1049	1323
	NS2	333	1017	1247	429	1023	1297
	KL2	303	987	1229	399	993	1267
AdvanceSchedule	SW	293	977	1209	391	999	1257
	NS	255	939	1183	353	973	1219
	KL	225	903	1147	317	949	1183
	SW2	281	967	1197	379	973	1247
	NS2	255	941	1171	353	947	1221
	KL2	225	905	1135	317	911	1185
StartSchedule		183	183	183	195	183	183
StopSchedule		137	137	137	137	137	137
GetScheduleStatus		181	181	181	181	181	181
GetScheduleValue		161	161	161	161	173	161
GetScheduleNext		71	71	71	71	71	71
SetScheduleNext		81	81	81	81	81	81
GetArrivalpointDelay		121	121	121	121	121	121
SetArrivalpointDelay		123	123	123	123	123	123
GetArrivalpointTasksetRef		53	53	53	53	53	53
GetArrivalpointNext		63	63	63	63	63	63
SetArrivalpointNext		69	69	69	69	69	69
TestArrivalpointWritable		73	73	73	73	73	73
GetExecutionTime		37	37	49	37	37	49
GetLargestExecutionTime		61	63	63	63	63	63
ResetLargestExecutionTime		37	39	39	39	39	39
GetStackOffset		69	69	69	69	69	69

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	169	273	367	181	255	405
	NS	155	259	353	167	241	391
	KL	125	225	321	137	211	363
TerminateTask	LExt	0	0	0	0	0	0
	H	843	853	867	853	853	867
ChainTask	SWL	1115	1253	1405	1231	1289	1531
	SWH	1205	1339	1489	1323	1375	1615
	NSL	1115	1253	1405	1231	1289	1531
	NSH	1203	1325	1475	1309	1361	1601
Schedule	SW	141	145	169	145	145	169
GetTaskID		93	95	95	95	95	95
GetTaskState		157	159	157	181	179	181
EnableAllInterrupts		39	39	39	39	39	39
DisableAllInterrupts		61	73	73	61	73	73
ResumeAllInterrupts		49	49	49	49	49	49
SuspendAllInterrupts		77	77	77	77	77	77
ResumeOSInterrupts		49	49	49	49	49	49
SuspendOSInterrupts		77	77	77	77	77	77
GetResource	Task	109	97	103	115	103	109
	Combined	115	115	115	121	121	121
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	125	125	125	129	129	129
	Combined	143	143	143	149	149	149
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	215	215	201
	NS	n/a	n/a	n/a	203	203	201
	KL	n/a	n/a	n/a	161	161	161
ClearEvent		n/a	n/a	n/a	99	99	99
GetEvent		n/a	n/a	n/a	73	73	73
WaitEvent	<default>	n/a	n/a	n/a	1285	1257	1409
	fp	n/a	n/a	n/a	1305	1275	1429
GetAlarmBase		213	227	227	213	213	213
GetAlarm		189	195	195	189	189	189

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		289	299	299	289	289	289
SetAbsAlarm		259	269	269	259	259	259
CancelAlarm		121	123	123	121	121	121
InitCounter		161	167	167	161	161	161
GetCounterValue		161	167	167	161	161	161
osek_tick_alarm	<default>	223	225	225	223	223	223
	KL	167	169	169	167	167	167
osek_incr_counter		37	39	39	37	37	37
GetActiveApplicationMode		13	13	13	13	13	13
StartOS		3507	3509	3509	3511	3511	3511
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	65	65	65	65	65	65
InitCOM		31	31	31	31	31	31
CloseCOM		31	31	31	31	31	31
StartCOM		105	105	105	429	429	429
StopCOM		65	65	65	65	65	65
ReadFlag		n/a	n/a	n/a	21	21	21
ResetFlag		n/a	n/a	n/a	15	15	15
ReceiveMessage		119	123	123	535	535	535
GetMessageResource		n/a	n/a	n/a	213	213	213
ReleaseMessageResource		n/a	n/a	n/a	221	221	221
GetMessageStatus		n/a	n/a	n/a	127	127	127
SendMessage	SW	321	435	529	725	799	949
	NS	301	415	509	717	791	941
	KL	237	347	443	647	721	873
ActivateTaskset	SW	129	729	959	141	737	1013
	NS	115	715	945	127	723	999
	KL	85	685	915	97	693	969
	SW2	129	729	959	141	737	1013
	NS2	115	715	945	127	723	999
	KL2	85	685	915	97	693	969
ChainTaskset	SWL	1087	1727	2013	1193	1779	2145
	SWH	1183	1825	2111	1289	1877	2243
	NSL	1087	1727	2013	1193	1779	2145
	NSH	1173	1811	2097	1279	1863	2229
GetTasksetRef		63	65	63	65	63	65

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		115	115	115	115	115	115
AssignTaskset		63	63	63	63	63	63
RemoveTaskset		119	119	119	119	119	119
TestSubTaskset		139	139	139	139	139	139
TestEquivalentTaskset		131	131	131	131	131	131
TickSchedule	SW	371	1049	1293	463	1085	1331
	NS	341	1023	1267	437	1059	1305
	KL	313	995	1239	409	1031	1277
	SW2	371	1051	1281	463	1059	1335
	NS2	341	1025	1255	437	1033	1309
	KL2	313	997	1227	409	1005	1281
AdvanceSchedule	SW	265	949	1193	363	985	1231
	NS	239	923	1167	337	959	1205
	KL	209	887	1131	301	923	1169
	SW2	265	951	1181	363	959	1235
	NS2	239	925	1155	337	933	1209
	KL2	209	889	1119	301	897	1173
StartSchedule		179	179	179	179	179	179
StopSchedule		133	133	133	133	133	133
GetScheduleStatus		185	185	185	185	185	185
GetScheduleValue		159	159	159	159	159	159
GetScheduleNext		69	69	69	69	69	69
SetScheduleNext		79	79	79	79	79	79
GetArrivalpointDelay		117	117	117	117	117	117
SetArrivalpointDelay		123	123	123	123	123	123
GetArrivalpointTasksetRef		53	53	53	53	53	53
GetArrivalpointNext		63	63	63	63	63	63
SetArrivalpointNext		69	69	69	69	69	69
TestArrivalpointWritable		73	73	73	73	73	73
GetExecutionTime		327	339	339	329	341	329
GetLargestExecutionTime		131	135	135	135	135	135
ResetLargestExecutionTime		109	113	113	113	113	113
GetStackOffset		69	69	69	69	69	69

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	675	799	891	701	781	937
	NS	751	891	981	789	871	1023
	KL	631	753	845	655	735	891
TerminateTask	LExt	907	919	933	919	919	933
	H	971	985	999	985	985	999
ChainTask	SWL	1719	1893	2043	1859	1929	2169
	SWH	1809	1979	2127	1951	2015	2255
	NSL	1813	1993	2141	1959	2027	2269
	NSH	1893	2069	2215	2041	2103	2345
Schedule	SW	215	221	245	221	221	245
GetTaskID		109	111	111	111	111	111
GetTaskState		699	729	727	737	735	737
EnableAllInterrupts		55	55	55	55	55	55
DisableAllInterrupts		77	77	77	77	77	89
ResumeAllInterrupts		77	77	77	77	77	77
SuspendAllInterrupts		93	93	93	93	93	93
ResumeOSInterrupts		77	77	77	77	77	77
SuspendOSInterrupts		93	93	93	93	93	93
GetResource	Task	1287	1355	613	1519	1513	733
	Combined	525	525	525	649	649	649
	CLEx	617	627	627	747	747	747
ReleaseResource	Task	575	575	575	703	703	703
	Combined	539	539	539	663	663	663
	CLEx	547	555	555	673	673	673
SetEvent	SW	n/a	n/a	n/a	775	775	761
	NS	n/a	n/a	n/a	807	807	805
	KL	n/a	n/a	n/a	733	733	731
ClearEvent		n/a	n/a	n/a	135	135	135
GetEvent		n/a	n/a	n/a	665	665	665
WaitEvent	<default>	n/a	n/a	n/a	1379	1349	1495
	fp	n/a	n/a	n/a	1399	1367	1515
GetAlarmBase		521	549	581	521	521	551
GetAlarm		497	517	549	497	497	527

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		731	771	803	731	731	761
SetAbsAlarm		657	693	725	657	657	687
CancelAlarm		429	445	477	429	429	459
InitCounter		807	869	923	807	807	855
GetCounterValue		439	457	457	439	439	439
osek_tick_alarm	<default>	241	243	243	241	241	241
	KL	167	169	169	167	167	167
osek_incr_counter		37	39	39	37	37	37
GetActiveApplicationMode		13	13	13	13	13	13
StartOS		3587	3589	3589	3591	3591	3591
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	73	73	73	73	73	73
InitCOM		31	31	31	31	31	31
CloseCOM		31	31	31	31	31	31
StartCOM		129	129	129	453	453	453
StopCOM		87	87	87	87	87	87
ReadFlag		n/a	n/a	n/a	97	97	97
ResetFlag		n/a	n/a	n/a	73	73	73
ReceiveMessage		349	361	361	765	765	765
GetMessageResource		n/a	n/a	n/a	1015	1015	1015
ReleaseMessageResource		n/a	n/a	n/a	995	995	995
GetMessageStatus		n/a	n/a	n/a	321	321	321
SendMessage	SW	1057	1199	1291	1487	1555	1711
	NS	1127	1285	1375	1569	1651	1803
	KL	977	1117	1209	1399	1479	1635
ActivateTaskset	SW	921	1655	1869	1019	1645	1935
	NS	985	1721	1935	1089	1711	2001
	KL	877	1611	1825	975	1601	1891
	SW2	921	1655	1869	1019	1645	1935
	NS2	985	1721	1935	1089	1711	2001
	KL2	877	1611	1825	975	1601	1891
ChainTaskset	SWL	2043	2829	3099	2263	2875	3255
	SWH	2139	2921	3191	2361	2967	3347
	NSL	2115	2915	3173	2337	2949	3329
	NSH	2201	2985	3255	2425	3031	3411
GetTasksetRef		583	613	611	613	611	613

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		231	231	231	231	231	231
AssignTaskset		171	171	171	171	171	171
RemoveTaskset		235	235	235	235	235	235
TestSubTaskset		245	245	245	245	245	245
TestEquivalentTaskset		237	237	237	237	237	237
TickSchedule	SW	513	2025	2257	1393	2079	2325
	NS	583	2097	2329	1465	2151	2397
	KL	453	1965	2197	1333	2019	2265
	SW2	513	2029	2243	1393	2019	2309
	NS2	583	2101	2315	1465	2091	2381
	KL2	453	1969	2183	1333	1959	2249
AdvanceSchedule	SW	419	1949	2181	1317	2003	2249
	NS	501	2025	2257	1393	2079	2325
	KL	377	1889	2121	1257	1943	2189
	SW2	419	1953	2167	1317	1943	2233
	NS2	501	2029	2243	1393	2019	2309
	KL2	377	1893	2107	1257	1883	2173
StartSchedule		299	299	299	299	297	299
StopSchedule		187	187	187	187	187	187
GetScheduleStatus		237	237	237	237	237	237
GetScheduleValue		211	211	211	211	211	211
GetScheduleNext		117	117	117	117	117	117
SetScheduleNext		181	181	181	181	181	181
GetArrivalpointDelay		193	193	193	193	193	193
SetArrivalpointDelay		239	239	239	239	239	239
GetArrivalpointTasksetRef		109	109	109	109	109	109
GetArrivalpointNext		119	119	119	119	119	119
SetArrivalpointNext		185	185	185	185	185	185
TestArrivalpointWritable		129	129	129	129	129	129
GetExecutionTime		361	373	373	363	363	363
GetLargestExecutionTime		619	649	649	649	649	649
ResetLargestExecutionTime		597	627	627	627	627	627
GetStackOffset		69	69	69	69	69	69

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	51	51	51	51	51	51
	Cat 2	61	179	179	179	179	179

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	51	51	51	51	51	51
	Cat 2	577	681	681	683	683	683

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	51	51	51	51	51	51
	Cat 2	577	681	681	683	683	683

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.

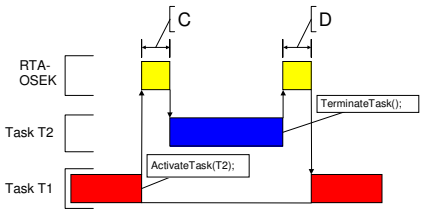

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

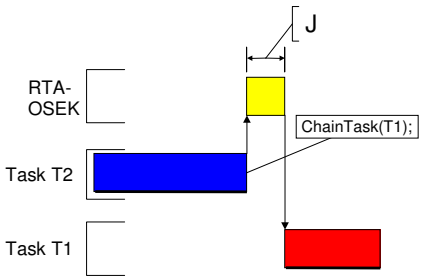

Figure 2: Task Chaining

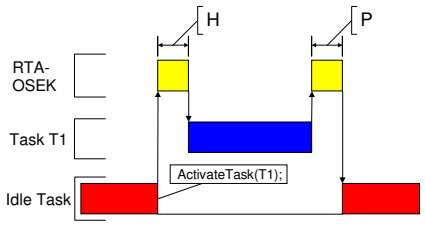

Figure 3: Task Activation from Idle Task

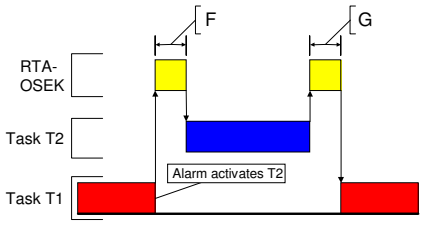

Figure 4: Task Activation from an Alarm

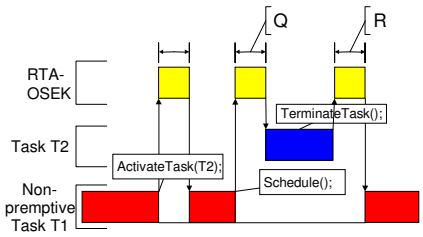

Figure 5: Non-Premptive Task Calls Schedule()

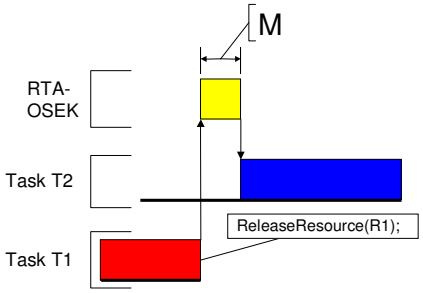

Figure 6: Blocked Task Activated by ReleaseResource()

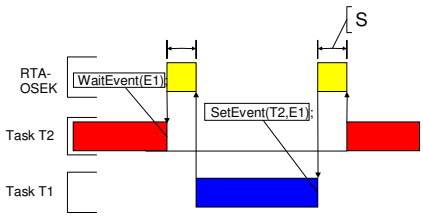

Figure 7: Waiting Task Activated by SetEvent()

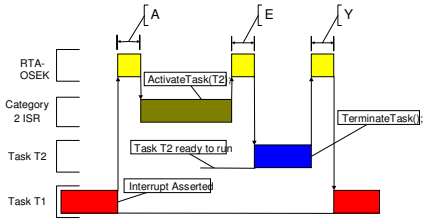

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	131	179	271	133	179	261
Figure 1: D	Heavy, Basic/Extended	243	277	351	317	309	399
ChainTask	Light, Basic	327	449	635	349	465	693
Figure 2: J	Heavy, Basic/Extended	835	987	1245	933	1035	1351
Pre-emption	Light, Basic	283	399	633	301	399	635
Figure 1: C	Heavy, Basic/Extended	395	519	681	495	551	789
From idle task	Light, Basic	283	399	633	301	399	635
Figure 3: H	Heavy, Basic/Extended	395	519	681	495	551	789
Triggered by alarm	Light, Basic	545	661	895	563	661	897
Figure 4: F	Heavy, Basic/Extended	657	781	943	757	813	1051
Schedule	Light, Basic	227	257	381	235	255	369
Figure 5: Q	Heavy, Basic/Extended	335	361	455	425	409	531
Release resource	Light, Basic	259	287	391	271	293	387
Figure 6: M	Heavy, Basic/Extended	367	391	465	461	447	549
SetEvent							

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	815	789	1093
From category 2 ISR	Light, Basic	197	341	445	319	341	435
Figure 8: E	Heavy, Basic/Extended	305	445	519	509	495	597

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	747	785	881	755	785	865
Figure 1: D	Heavy, Basic/Extended	843	873	943	913	899	999
ChainTask	Light, Basic	1021	1141	1317	1053	1159	1385
Figure 2: J	Heavy, Basic/Extended	2121	2263	2515	2221	2307	2635
Pre-emption	Light, Basic	793	903	1125	819	907	1133
Figure 1: C	Heavy, Basic/Extended	887	1017	1181	999	1057	1311
From idle task	Light, Basic	793	903	1125	819	907	1133
Figure 3: H	Heavy, Basic/Extended	887	1017	1181	999	1057	1311
Triggered by alarm	Light, Basic	1055	1165	1387	1081	1169	1395
Figure 4: F	Heavy, Basic/Extended	1149	1279	1443	1261	1319	1573
Schedule	Light, Basic	735	757	879	751	751	861
Figure 5: Q	Heavy, Basic/Extended	829	861	955	931	909	1037
Release resource	Light, Basic	765	781	879	785	791	877
Figure 6: M	Heavy, Basic/Extended	859	885	955	965	949	1053
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1277	1249	1571
From category 2 ISR	Light, Basic	1333	1457	1555	1455	1461	1547
Figure 8: E	Heavy, Basic/Extended	1427	1561	1631	1635	1619	1723

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	907	947	1045	919	947	1027
Figure 1: D	Heavy, Basic/Extended	971	1005	1073	1045	1029	1131
ChainTask	Light, Basic	1623	1779	1953	1679	1797	2021
Figure 2: J	Heavy, Basic/Extended	2851	3033	3281	2983	3079	3405
Pre-emption	Light, Basic	1291	1421	1641	1331	1425	1657
Figure 1: C	Heavy, Basic/Extended	1387	1537	1699	1513	1577	1837
From idle task	Light, Basic	1291	1421	1641	1331	1425	1657
Figure 3: H	Heavy, Basic/Extended	1387	1537	1699	1513	1577	1837
Triggered by alarm	Light, Basic	1571	1701	1921	1611	1705	1937
Figure 4: F	Heavy, Basic/Extended	1667	1817	1979	1793	1857	2117
Schedule	Light, Basic	797	821	943	815	813	923
Figure 5: Q	Heavy, Basic/Extended	893	927	1021	997	973	1101
Release resource	Light, Basic	1179	1195	1293	1321	1327	1413
Figure 6: M	Heavy, Basic/Extended	1275	1301	1371	1503	1487	1591
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1835	1807	2129
From category 2 ISR	Light, Basic	1367	1491	1589	1489	1495	1581
Figure 8: E	Heavy, Basic/Extended	1463	1597	1667	1671	1655	1759

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	20	20	23	20	20	23
BCC1 lightweight, floating-point	23	23	26	23	23	26
BCC1 heavyweight, integer	28	28	31	28	28	31
BCC1 heavyweight, floating-point	28	28	31	28	28	31
BCC2 lightweight, integer	n/a	23	28	n/a	23	28
BCC2 lightweight, floating-point	n/a	23	28	n/a	23	28
BCC2 heavyweight, integer	n/a	30	33	n/a	30	33
BCC2 heavyweight, floating-point	n/a	30	33	n/a	30	33
ECC1 heavyweight, integer	n/a	n/a	n/a	30	30	33
ECC1 heavyweight, floating-point	n/a	n/a	n/a	30	30	33
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	33
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	33
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	23	23	23	23	23	23
BCC1 lightweight, floating-point	26	26	26	26	26	26
BCC1 heavyweight, integer	31	31	31	31	31	31
BCC1 heavyweight, floating-point	31	31	31	31	31	31
BCC2 lightweight, integer	n/a	26	28	n/a	26	28
BCC2 lightweight, floating-point	n/a	26	28	n/a	26	28
BCC2 heavyweight, integer	n/a	33	33	n/a	33	33
BCC2 heavyweight, floating-point	n/a	33	33	n/a	33	33
ECC1 heavyweight, integer	n/a	n/a	n/a	33	33	33
ECC1 heavyweight, floating-point	n/a	n/a	n/a	33	33	33
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	33
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	33

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	29	29	32	29	29	32
BCC1 lightweight, floating-point	32	32	35	32	32	35
BCC1 heavyweight, integer	37	37	40	37	37	40
BCC1 heavyweight, floating-point	37	37	40	37	37	40
BCC2 lightweight, integer	n/a	32	37	n/a	32	37
BCC2 lightweight, floating-point	n/a	32	37	n/a	32	37
BCC2 heavyweight, integer	n/a	39	42	n/a	39	42
BCC2 heavyweight, floating-point	n/a	39	42	n/a	39	42
ECC1 heavyweight, integer	n/a	n/a	n/a	39	39	42
ECC1 heavyweight, floating-point	n/a	n/a	n/a	39	39	42
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	42
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	42
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	32	32	32	32	32	32
BCC1 lightweight, floating-point	35	35	35	35	35	35
BCC1 heavyweight, integer	40	40	40	40	40	40
BCC1 heavyweight, floating-point	40	40	40	40	40	40
BCC2 lightweight, integer	n/a	35	37	n/a	35	37
BCC2 lightweight, floating-point	n/a	35	37	n/a	35	37
BCC2 heavyweight, integer	n/a	42	42	n/a	42	42
BCC2 heavyweight, floating-point	n/a	42	42	n/a	42	42
ECC1 heavyweight, integer	n/a	n/a	n/a	42	42	42
ECC1 heavyweight, floating-point	n/a	n/a	n/a	42	42	42
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	42
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	42

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	29	29	32	29	29	32
BCC1 lightweight, floating-point	32	32	35	32	32	35
BCC1 heavyweight, integer	37	37	40	37	37	40
BCC1 heavyweight, floating-point	37	37	40	37	37	40
BCC2 lightweight, integer	n/a	32	37	n/a	32	37
BCC2 lightweight, floating-point	n/a	32	37	n/a	32	37
BCC2 heavyweight, integer	n/a	39	42	n/a	39	42
BCC2 heavyweight, floating-point	n/a	39	42	n/a	39	42
ECC1 heavyweight, integer	n/a	n/a	n/a	39	39	42
ECC1 heavyweight, floating-point	n/a	n/a	n/a	39	39	42
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	42
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	42
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	32	32	32	32	32	32
BCC1 lightweight, floating-point	35	35	35	35	35	35
BCC1 heavyweight, integer	40	40	40	40	40	40
BCC1 heavyweight, floating-point	40	40	40	40	40	40
BCC2 lightweight, integer	n/a	35	37	n/a	35	37
BCC2 lightweight, floating-point	n/a	35	37	n/a	35	37
BCC2 heavyweight, integer	n/a	42	42	n/a	42	42
BCC2 heavyweight, floating-point	n/a	42	42	n/a	42	42
ECC1 heavyweight, integer	n/a	n/a	n/a	42	42	42
ECC1 heavyweight, floating-point	n/a	n/a	n/a	42	42	42
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	42
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	42

5 Compatibility with Pre-v5 Kernels

5.1 Updating the Application Version

To convert an existing v3.x OIL configuration file to v5.0x (i.e. v5.00 or v5.01), load the file into the v5 RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.0x. When the OIL configuration file is saved it will then use the v5.0x format and the v5.0x kernel libraries. This process can be reversed to move back to earlier kernel versions.

5.2 32 Bit Timer Drivers

The v3.x kernel uses sixteen bit timer values, whereas the v5.0x kernels use thirty-two bit timer values. Therefore any existing applications' timer drivers will need modifying. Since the S12X Timer Module provides only sixteen bit timer registers the upper sixteen bits will need to be emulated in software. The provided example application demonstrates one method of achieving this for the `TCNT` timer register.

5.3 Memory Model

The v3.x kernel was build for Unbanked memory model whereas the V5.0x kernels are built for Banked memory model (all Function pointers are 32 bit and data pointers are 16 bit) with compiler option +modf which makes all functions as @far.

5.4 Data Initialization

Data initialized by compiler Startup code and StartOS() API are now contained in different sections. The new sections os_pur and os_pir2 have 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. Existing linker command files will require modification as demonstrated by the example application.

6 Compatibility with V5 Kernels

6.1 Updating the Application Version

To convert an existing v5.00 OIL configuration file to v5.01, load the file into the v5 RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.01. When the OIL configuration file is saved it will then use v5.01 kernel libraries. This process can be reversed to move back to earlier kernel versions.

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.