RTA-OSEK

Binding Manual: 16LX/FUJITSU

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

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

www.etas.de

Japan

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

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

www.etas.co.jp

Korea

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

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

www.etas.co.kr

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

France

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

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

www.etas.fr

Great Britain

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

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

www.etas-uk.net

Copyright Notice

Version: M00083-002

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 16LX/FUJITSU port of LiveDevices' RTA-OSEK. It supplements the more general information in the *RTA-OSEK User Guide*.

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

1.1 Who Should Read this Guide?

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

1.2 Conventions

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

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

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

2 Toolchain Issues

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

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

The RTA-OSEK Component's libraries are built with the medium memory model. All application programs should also be built to use the medium memory model. If you wish to use any of the other three memory models you should contact LiveDevices.

2.1 Compiler

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

Vendor	Fujitsu
Compiler	F2MC-16 Family Softune C compiler
Version	V30L15 (Workbench version V30L33)

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

Option	Description
`-cpu %CPU_TYPE%`	Select target CPU
`-model MEDIUM`	Medium memory model

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

Option	Description
`-ramconst`	Specifies that the mirror function will not be used

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

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

Option	Description
`-cpu %CPU_TYPE%`	Select target CPU
`-model MEDIUM`	Medium memory model

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

Option	Description
`-ramconst`	Specifies that the mirror function will not be used

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. MB90F548G).

2.2 Assembler

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

Vendor	Fujitsu
Assembler	F2MC-16 Family Softune Assembler
Version	V30L12 (Workbench version V30L33)

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

Option	Description
`-cpu %CPU_TYPE%`	Select target CPU

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

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

Option	Description
`-cpu %CPU_TYPE%`	Select target CPU

2.3 Linker/Locator

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

Option	Description
`-sc os_pird+os_pid/const/word=0xff4000`	Place os_pird and os_pid in ROM mirror section

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

Sections	ROM/RAM	Description
`os_pir`	RAM	RTA-OSEK initialized data
`os_pur`	RAM	RTA-OSEK uninitialized data

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

C Library Functions	Description
`setjmp()`	Softune library `setjmp` routine
`longjmp()`	Softune library `longjmp` routine

The os_pird and os_pid sections must be placed in the "ROM mirror" section so that they are visible to the 16-bit data pointers that access data in the medium memory model.

2.4 Debugger

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

At the time of writing, we were not aware of any debuggers for the Fujitsu 16Lx and Fx with support for ORTI.

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

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

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for 16LX/FUJITSU. 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 *Fujitsu hardware reference manuals*.

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

Interrupt Priority Level (IPL)	**ILM register**	**I bit in PS register**	**Description**
0	7	1	User level
1	6	1	Category 1 and 2 interrupts
2	5	1	Category 1 and 2 interrupts
3	4	1	Category 1 and 2 interrupts
4	3	1	Category 1 and 2 interrupts
5	2	1	Category 1 and 2 interrupts
6	1	1	Category 1 and 2 interrupts
7	0	1	Category 1 and 2 interrupts
8	any	0	Category 1 software interrupt only

3.1.2 Interrupt Vectors

On the Fujitsu 16LX and 16FX, vectors are aligned on four byte boundaries between 0xFFFC00 and 0xFFFFFC. The 16FX, however, allows the vector table to be re-located to other locations and this process is described in further Section. RTA-OSEK allows ISRs to be bound to any vector, subject to the restrictions on ISRs described in Section 3.1.3

Important: Extended intelligent I/O (see Fujitsu hardware documentation) should only be used with Category 1 ISRs.

3.1.3 Interrupt Priority Levels

The priority at which a hardware interrupt is taken is set in the ICR hardware registers for the 16LX and the ILR, IDX registers on the 16FX. On the 16LX each ICR register applies to two peripheral devices (e.g. on the MB90F548G chip, ICR03 sets the priority for the "16-bit reload timer 0" interrupt and for the "A/D converter" interrupt). This means that two devices attached to a single ICR share a single hardware interrupt priority level. On the 16FX however, each ICR register applies to one interrupt vector and corresponding ILR register, via an IDX register to set its priority.

The RTA-OSEK GUI generates a table, called `os_InitIrqLevels`, which can be used to initialize the ICR (ILR, IDX) registers. This table contains the priority levels for interrupts defined in the application.

Important: Either the `os_InitIrqLevels` table or equivalent values must be used to initialize the ICR (ILR, IDX) registers before the call to `StartOS()` otherwise interrupts will not work correctly in the application.

The `init_target()` function in `target.c` in the example application (located in `<RTA-OSEK install directory>\FUJI16LX\Example\`) for the 16LX and (located in `<RTA-OSEK install directory>\FUJI16LX\Example16FX\`) for the 16FX gives an example of how to copy `os_InitIrqLevels` to the correct location.

If the user wishes to initialize the ICR (ILR, IDX) registers directly and does not require the `os_InitIrqLevels` table then the RTA-OSEK generated file `osekdefs.c` can be compiled with the macro `OS_NO_ICL_INIT` defined. The `os_InitIrqLevels` table will then not be contained in the final application.

On the 16LX ICR sharing by interrupt sources has ramifications with respect to interrupt sources that are not explicitly bound to an ISR. When one of the interrupt sources on an ICR is bound to an ISR and the other is not, the priority of the unbound source is forced by the hardware to be the same as the bound one. When neither interrupt source on an ICR is bound to an ISR, the value in the ICR is set to effectively disable the interrupts.

Note: If a default interrupt shares an ICR with another ISR then that default interrupt (but not other default interrupts) will trigger at the level of the other ISR value.

If Category 1 interrupts are triggered from peripheral interrupt sources, they must have a priority (IPL) between 1 and 7 (where 1 is the lowest and 7 is the highest). All Category 1 ISRs must have a priority greater than or equal to that of the highest Category 2 interrupt.

Important: If you define a Category 1 interrupt at level 8, you must never trigger the interrupt using a hardware source.

Category 2 interrupts can have priorities (IPLs) between 1 and 7 (where 1 is the lowest and 7 is the highest).

3.1.4 Software Interrupts

All software interrupts must be Category 1 and priority (IPL) 8. Vectors that can be used for peripheral interrupt sources can also be used for software interrupts. However, for a software interrupt, the priority in the ICR (ILR, IDX) corresponding to that vector is meaningless. Therefore, in the case where a peripheral interrupt source and a software interrupt have vectors that share the same ICR, it is permitted to have their ISRs at different priorities.

3.1.5 Category 1 Handlers

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

3.1.6 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.

3.1.7 Vector Table Issues

When you configure your application with the RTA-OSEK GUI you can choose whether or not a vector table is generated within `osgen.asm`. Note that this generated vector table includes 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 (VVVVVV represents the 6 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
`0xVVVVVV`	`_os_wrapper_VVVVVV`
`eg :0xFFFF6C`	`_os_wrapper_FFFF6C`

3.1.8 Automatic vector table generation

The build process will generate a vector table covering all Category 1 and Category 2 ISRs defined in the RTA-OSEK GUI.

The reset vector (at address 0xFFFFDC) is set to the label `_start`.

If a default interrupt is specified, a vector table covering all vectors will be generated. If the default interrupt is not specified a vector table will be generated that starts at the lowest used vector.

3.1.9 Manual vector table generation

For each configured ISR, its associated vector must be programmed with the address of its handler. Other vectors may be programmed with the address of a default interrupt handler, if present.

For instance, the following example will place the address of `os_wrapper_FFFF6C` on interrupt vector number 36.

```
#pragma intvect os_wrapper_FFFF6C 36
```

3.1.10 Re-locatable vector table

The 16FX has the ability to relocate the vector table to any suitable location, in steps of 1 kBytes by use of the TBR register. Upon reset this register is loaded with 0xFFFC, thus giving a vector table start address of 0xFFFC00.

Note: Use of TBR = 0x0000 is not recommended

The most straight forward way of implementing a relocated vector table, is through the use of the manual vector table generation method. Therefore, in order to relocate the vector table to, for example, a start address of 0xFE0000, then at a point in your code before interrupts and their associated priorities are initialized, add:

```
TBR = 0xFE00;
```

In a suitable header file add:

```
#pragma section intvect=intvect_relocated,
locate=0xFE0000
```

It is important that the value of the TBR register and relocated vector table match. Further information regarding a re-locatable vector table can be found in the 16FX datasheet and the #pragma directive in the Softune F^{2}MC-16 Family C Compiler manual.

3.2 Register Settings

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

Register	**Required Value**
SSP	Start of the system stack

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

Reserved Registers And Bits	**Notes**
PS	The Processor Status register (including the ILM, the RP and the CCR) should not be altered directly by user code.

Note: Instructions that indirectly change the condition codes in the CCR can, of course, be used freely.

The RTA-OSEK Component only uses register bank zero (i.e. RP=0 in PS) and this should not be altered by user code. Additional register bank memory can be used for other application purposes.

3.3 Stack Usage

3.3.1 Number of Stacks

The RTA-OSEK Component uses only the system stack and expects the user code to do the same.

The first argument to `StackFaultHook` is always 0.

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

Note the RTA-OSEK Component requires a label "_systemstack_top" marking the top of the stack. An example of how to place this label can be found in `start.asm` in the example application.

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): 116

Timing

API max usage (bytes): 116

Extended

API max usage (bytes): 84

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.

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	**Fujitsu 16LX/MB96F346YSA**
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	116	116	120	189	189	193
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	182	182	186	255	255	259
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	52	52	52	52	52	52
	ROM	206	206	210	279	279	283
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	30	34	n/a	30	34
ECC1, Integer task	RAM	n/a	n/a	n/a	26	26	26
	ROM	n/a	n/a	n/a	36	36	36
ECC1, floating-point task	RAM	n/a	n/a	n/a	27	27	27
	ROM	n/a	n/a	n/a	36	36	36
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	28
	ROM	n/a	n/a	n/a	n/a	n/a	40
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	29
	ROM	n/a	n/a	n/a	n/a	n/a	40
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	34	34	34	34	34	34
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	42	42	42	42	42	42
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	32	32	32	32	32	32
Counter	RAM	4	4	4	4	4	4
	ROM	71	71	71	71	71	71
Message	RAM	11	11	11	51	51	51
	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	78	78	78	78	78	78
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	38	42	n/a	38	42
ECC1, Integer task	RAM	n/a	n/a	n/a	36	36	36
	ROM	n/a	n/a	n/a	44	44	44
ECC1, floating-point task	RAM	n/a	n/a	n/a	37	37	37
	ROM	n/a	n/a	n/a	44	44	44

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	38
	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	39
	ROM	n/a	n/a	n/a	n/a	n/a	48
Category 2 ISR	RAM	10	10	10	10	10	10
	ROM	50	50	50	50	50	50
Category 2 ISR, floating-point	RAM	11	11	11	11	11	11
	ROM	58	58	58	58	58	58
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	32	32	32	32	32	32
Counter	RAM	4	4	4	4	4	4
	ROM	71	71	71	71	71	71
Message	RAM	11	11	11	51	51	51
	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	78	78	78	78	78	78
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
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

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	11	11	11	11	11	11
	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	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	46	46	46
ECC1, floating-point task	RAM	n/a	n/a	n/a	39	39	39
	ROM	n/a	n/a	n/a	46	46	46
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	40
	ROM	n/a	n/a	n/a	n/a	n/a	50
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	41
	ROM	n/a	n/a	n/a	n/a	n/a	50
Category 2 ISR	RAM	11	11	11	11	11	11
	ROM	54	54	54	54	54	54
Category 2 ISR, floating-point	RAM	12	12	12	12	12	12
	ROM	62	62	62	62	62	62
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Linked resource	RAM	4	4	4	4	4	4
	ROM	14	14	14	14	14	14
Alarm	RAM	10	10	10	10	10	10
	ROM	34	34	34	34	34	34
Counter	RAM	4	4	4	4	4	4
	ROM	73	73	73	73	73	73
Message	RAM	11	11	11	51	51	51
	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	78	78	78	78	78	78
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.

Variant	Description
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.
Task	Resource is used only by tasks.

Standard

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
Service name	Variant	Notes						
ActivateTask	SW	1	100	153	200	104	157	219
	NS		81	134	181	85	138	200
	KL	2	61	115	162	65	119	181
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	22	22	22	22	22	22
ChainTask	SWL	1, 8	80	135	182	84	139	201
	SWH	1, 9	104	157	205	108	161	224
	NSL	8	80	135	182	84	139	201
	NSH	9	98	151	199	102	155	218
Schedule			55	55	79	55	55	79
GetTaskID			21	21	21	21	21	21
GetTaskState			57	57	57	72	72	72
EnableAllInterrupts			10	10	10	10	10	10
DisableAllInterrupts			10	10	10	10	10	10
ResumeAllInterrupts			26	26	26	26	26	26
SuspendAllInterrupts			20	20	20	20	20	20
ResumeOSInterrupts			28	28	28	28	28	28
SuspendOSInterrupts			35	35	35	35	35	35
GetResource	Task	7	26	26	30	26	26	30
	Combined	6	63	63	63	63	63	63
	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	101	101	101	101	101	101
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	84	84	170
	NS		n/a	n/a	n/a	65	65	148
	NS1i	10	n/a	n/a	n/a	33	n/a	n/a

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	KL	2	n/a	n/a	n/a	52	52	136
	KL1i	2, 10	n/a	n/a	n/a	17	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	17	17	17
WaitEvent	<default>		n/a	n/a	n/a	181	181	345
	fp	11	n/a	n/a	n/a	213	213	412
	1i	10	n/a	n/a	n/a	18	n/a	n/a
GetAlarmBase			41	41	41	41	41	41
GetAlarm			83	83	83	83	83	83
SetRelAlarm			644	644	644	644	644	644
SetAbsAlarm			727	727	727	727	727	727
CancelAlarm			64	64	64	64	64	64
InitCounter			59	59	59	59	59	59
GetCounterValue			73	73	73	73	73	73
GetScheduleTableStatus		34	62	84	84	62	84	84
NextScheduleTable		34	75	206	206	75	206	206
StartScheduleTable		34	112	171	171	112	171	171
StopScheduleTable		34	71	99	99	71	99	99
ScheduleTable expiry point	ActivateTask		10	10	10	10	10	10
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	13	13	13
ScheduleTable expiry point	Callback		1	1	1	1	1	1
ScheduleTable expiry point	Tick counter		13	13	13	13	13	13
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	23	23	23	23	23	23
Process container	Non-Yielding	33	9	9	9	9	9	9
osek_tick_alarm	<default>		63	63	63	63	63	63
	KL	2	49	49	49	49	49	49
osek_incr_counter			54	54	54	54	54	54
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			88	88	88	88	88	88
ShutdownOS	NoHook	12	17	17	17	17	17	17
	Hook	13	29	29	29	29	29	29
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			17	17	17	17	17	17
StopCOM			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	
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	142	142	142
	CCCB	15	142	142	142	142	142	142
GetMessageResource			34	34	34	34	34	34
ReleaseMessageResource			34	34	34	34	34	34
GetMessageStatus			55	55	55	55	55	55
SendMessage	SW CCCA	1, 14	72	72	72	178	178	178
	SW CCCB	1, 15	162	162	162	178	178	178
	NS CCCA	14	72	72	72	178	178	178
	NS CCCB	15	162	162	162	178	178	178
	KL CCCA	2, 14	56	56	56	166	166	166
	KL CCCB	2, 15	150	150	150	166	166	166
main_dispatch	NoHook	12	75	75	108	75	75	108
	Hook	13	106	106	139	106	106	139
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	65	65	65	65	65	65
	B1HF	21	73	73	73	73	73	73
	B2LI	22	n/a	51	79	n/a	51	79
	B2LF	23	n/a	59	87	n/a	59	87
	B2HI	24	n/a	178	265	n/a	178	265
	B2HF	25	n/a	186	273	n/a	186	273
	E1HI	26	n/a	n/a	n/a	278	278	366
	E1HF	27	n/a	n/a	n/a	286	286	374
	E2HI	28	n/a	n/a	n/a	n/a	n/a	366
	E2HF	29	n/a	n/a	n/a	n/a	n/a	374
ErrorHook support		16	26	26	26	26	26	26
	ServiceID	17	32	32	32	32	32	32
	Parameters	18	53	53	53	53	53	53
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	94	195	247	102	214	281
	NS		75	176	228	83	194	262
	KL	2	56	152	204	64	169	237
ChainTaskset	SWL	1, 8	81	191	243	87	202	273
	SWH	1, 9	113	235	286	119	246	316

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	NSL	8	81	191	243	87	202	273
	NSH	9	107	229	280	113	240	310
GetTasksetRef			13	13	13	13	13	13
MergeTaskset			36	36	36	36	36	36
AssignTaskset			13	13	13	13	13	13
RemoveTaskset			37	37	37	37	37	37
TestSubTaskset			46	46	46	46	46	46
TestEquivalentTaskset			41	41	41	41	41	41
TickSchedule	SW	1	166	153	153	153	153	153
	NS		147	134	134	134	134	134
	KL	2	135	120	120	120	120	120
AdvanceSchedule	SW	1	157	140	140	140	140	140
	NS		138	119	119	119	119	119
	KL	2	124	105	105	105	105	105
StartSchedule			69	69	69	69	69	69
StopSchedule			42	42	42	42	42	42
GetScheduleStatus			75	75	75	75	75	75
GetScheduleValue			56	56	56	56	56	56
GetScheduleNext			15	15	15	15	15	15
SetScheduleNext			18	18	18	18	18	18
GetArrivalpointDelay			20	20	20	20	20	20
SetArrivalpointDelay			14	14	14	14	14	14
GetArrivalpointTasksetRef			11	11	11	11	11	11
GetArrivalpointNext			14	14	14	14	14	14
SetArrivalpointNext			12	12	12	12	12	12
TestArrivalpointWritable			26	26	26	26	26	26
GetExecutionTime			3	3	3	3	3	3
GetLargestExecutionTime			16	16	16	16	16	16
ResetLargestExecutionTime			2	2	2	2	2	2
GetStackOffset			18	18	18	18	18	18

Timing

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	100	153	200	104	157	219
	NS		81	134	181	85	138	200
	KL	2	61	115	162	65	119	181
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	22	22	22	22	22	22
ChainTask	SWL	1, 8	80	135	182	84	139	201
	SWH	1, 9	104	157	205	108	161	224
	NSL	8	80	135	182	84	139	201
	NSH	9	98	151	199	102	155	218
Schedule			67	67	91	67	67	91
GetTaskID			21	21	21	21	21	21
GetTaskState			57	57	57	72	72	72
EnableAllInterrupts			10	10	10	10	10	10
DisableAllInterrupts			10	10	10	10	10	10
ResumeAllInterrupts			26	26	26	26	26	26
SuspendAllInterrupts			20	20	20	20	20	20
ResumeOSInterrupts			28	28	28	28	28	28
SuspendOSInterrupts			35	35	35	35	35	35
GetResource	Task	7	26	26	30	26	26	30
	Combined	6	63	63	63	63	63	63
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	61	61	61	61	61	61
	Combined	6	125	125	125	125	125	125
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	84	84	170
	NS		n/a	n/a	n/a	65	65	148
	NS1i	10	n/a	n/a	n/a	33	n/a	n/a
	KL	2	n/a	n/a	n/a	52	52	136
	KL1i	2, 10	n/a	n/a	n/a	17	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	17	17	17
WaitEvent	<default>		n/a	n/a	n/a	224	224	388
	fp	11	n/a	n/a	n/a	256	256	455

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	65	n/a	n/a
GetAlarmBase			41	41	41	41	41	41
GetAlarm			83	83	83	83	83	83
SetRelAlarm			644	644	644	644	644	644
SetAbsAlarm			727	727	727	727	727	727
CancelAlarm			64	64	64	64	64	64
InitCounter			59	59	59	59	59	59
GetCounterValue			73	73	73	73	73	73
GetScheduleTableStatus		34	62	84	84	62	84	84
NextScheduleTable		34	75	206	206	75	206	206
StartScheduleTable		34	112	171	171	112	171	171
StopScheduleTable		34	71	99	99	71	99	99
ScheduleTable expiry point	ActivateTask		10	10	10	10	10	10
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	13	13	13
ScheduleTable expiry point	Callback		1	1	1	1	1	1
ScheduleTable expiry point	Tick counter		13	13	13	13	13	13
ScheduleTable expiry point	Final		28	28	28	28	28	28
GetISRID		4	28	28	28	28	28	28
Process container	Yielding	32	23	23	23	23	23	23
Process container	Non-Yielding	33	9	9	9	9	9	9
osek_tick_alarm	<default>		63	63	63	63	63	63
	KL	2	49	49	49	49	49	49
osek_incr_counter			54	54	54	54	54	54
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			124	124	124	124	124	124
ShutdownOS	NoHook	12	17	17	17	17	17	17
	Hook	13	29	29	29	29	29	29
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			17	17	17	17	17	17
StopCOM			12	12	12	12	12	12
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	142	142	142
	CCCB	15	142	142	142	142	142	142
GetMessageResource			34	34	34	34	34	34
ReleaseMessageResource			34	34	34	34	34	34

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			55	55	55	55	55	55
SendMessage	SW CCCA	1, 14	72	72	72	178	178	178
	SW CCCB	1, 15	162	162	162	178	178	178
	NS CCCA	14	72	72	72	178	178	178
	NS CCCB	15	162	162	162	178	178	178
	KL CCCA	2, 14	56	56	56	166	166	166
	KL CCCB	2, 15	150	150	150	166	166	166
main_dispatch	NoHook	12	152	152	187	152	152	187
	Hook	13	176	176	219	176	176	219
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	80	80	80	80	80	80
	B1HF	21	88	88	88	88	88	88
	B2LI	22	n/a	40	70	n/a	40	70
	B2LF	23	n/a	48	78	n/a	48	78
	B2HI	24	n/a	189	276	n/a	189	276
	B2HF	25	n/a	197	284	n/a	197	284
	E1HI	26	n/a	n/a	n/a	317	317	405
	E1HF	27	n/a	n/a	n/a	325	325	413
	E2HI	28	n/a	n/a	n/a	n/a	n/a	405
	E2HF	29	n/a	n/a	n/a	n/a	n/a	413
ErrorHook support		16	26	26	26	26	26	26
	ServiceID	17	32	32	32	32	32	32
	Parameters	18	53	53	53	53	53	53
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	54	54	54	54	54	54
Timing_termination		4	108	108	108	108	108	108
ActivateTaskset	SW	1	94	195	247	102	214	281
	NS		75	176	228	83	194	262
	KL	2	56	152	204	64	169	237
ChainTaskset	SWL	1, 8	81	191	243	87	202	273
	SWH	1, 9	113	235	286	119	246	316
	NSL	8	81	191	243	87	202	273
	NSH	9	107	229	280	113	240	310
GetTasksetRef			13	13	13	13	13	13
MergeTaskset			36	36	36	36	36	36
AssignTaskset			13	13	13	13	13	13
RemoveTaskset			37	37	37	37	37	37

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestSubTaskset			46	46	46	46	46	46
TestEquivalentTaskset			41	41	41	41	41	41
TickSchedule	SW	1	166	153	153	153	153	153
	NS		147	134	134	134	134	134
	KL	2	135	120	120	120	120	120
AdvanceSchedule	SW	1	157	140	140	140	140	140
	NS		138	119	119	119	119	119
	KL	2	124	105	105	105	105	105
StartSchedule			69	69	69	69	69	69
StopSchedule			42	42	42	42	42	42
GetScheduleStatus			75	75	75	75	75	75
GetScheduleValue			56	56	56	56	56	56
GetScheduleNext			15	15	15	15	15	15
SetScheduleNext			18	18	18	18	18	18
GetArrivalpointDelay			20	20	20	20	20	20
SetArrivalpointDelay			14	14	14	14	14	14
GetArrivalpointTasksetRef			11	11	11	11	11	11
GetArrivalpointNext			14	14	14	14	14	14
SetArrivalpointNext			12	12	12	12	12	12
TestArrivalpointWritable			26	26	26	26	26	26
GetExecutionTime			64	64	64	64	64	64
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			21	21	21	21	21	21
GetStackOffset			18	18	18	18	18	18

Extended

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	177	227	276	183	231	299
	NS		227	283	331	233	287	354
	KL	2	124	178	227	130	182	249
TerminateTask	LExt	3	82	82	82	82	82	82

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	107	107	107	107	107	107
ChainTask	SWL	1, 8	203	260	307	209	265	327
	SWH	1, 9	232	287	335	238	291	355
	NSL	8	271	330	377	278	334	396
	NSH	9	292	350	398	298	355	418
Schedule			158	158	184	158	158	184
GetTaskID			29	29	29	29	29	29
GetTaskState			175	175	175	181	181	181
EnableAllInterrupts			18	18	18	18	18	18
DisableAllInterrupts			20	20	20	20	20	20
ResumeAllInterrupts			63	63	63	63	63	63
SuspendAllInterrupts			30	30	30	30	30	30
ResumeOSInterrupts			65	65	65	65	65	65
SuspendOSInterrupts			45	45	45	45	45	45
GetResource	Task	7	315	315	270	315	315	270
	Combined	6	269	269	269	269	269	269
	CLEx	3	258	258	258	258	258	258
ReleaseResource	Task	7	277	277	277	277	277	277
	Combined	6	334	334	334	334	334	334
	CLEx	3	255	255	255	255	255	255
SetEvent	SW	1	n/a	n/a	n/a	214	214	301
	NS		n/a	n/a	n/a	270	270	355
	NS1i	10	n/a	n/a	n/a	162	n/a	n/a
	KL	2	n/a	n/a	n/a	165	165	252
	KL1i	2, 10	n/a	n/a	n/a	129	n/a	n/a
ClearEvent			n/a	n/a	n/a	93	93	93
GetEvent			n/a	n/a	n/a	124	124	124
WaitEvent	<default>		n/a	n/a	n/a	318	318	468
	fp	11	n/a	n/a	n/a	350	350	535
	1i	10	n/a	n/a	n/a	158	n/a	n/a
GetAlarmBase			130	130	130	130	130	130
GetAlarm			137	137	137	137	137	137
SetRelAlarm			818	818	818	818	818	818
SetAbsAlarm			922	922	922	922	922	922
CancelAlarm			119	119	119	119	119	119
InitCounter			195	195	195	195	195	195
GetCounterValue			150	150	150	150	150	150

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	72	94	94	72	94	94
NextScheduleTable		34	85	216	216	85	216	216
StartScheduleTable		34	122	181	181	122	181	181
StopScheduleTable		34	81	109	109	81	109	109
ScheduleTable expiry point	ActivateTask		10	10	10	10	10	10
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	13	13	13
ScheduleTable expiry point	Callback		1	1	1	1	1	1
ScheduleTable expiry point	Tick counter		13	13	13	13	13	13
ScheduleTable expiry point	Final		28	28	28	28	28	28
GetISRID		4	38	38	38	38	38	38
Process container	Yielding	32	23	23	23	23	23	23
Process container	Non-Yielding	33	9	9	9	9	9	9
osek_tick_alarm	<default>		82	82	82	82	82	82
	KL	2	49	49	49	49	49	49
osek_incr_counter			54	54	54	54	54	54
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			134	134	134	134	134	134
ShutdownOS	NoHook	12	22	22	22	22	22	22
	Hook	13	34	34	34	34	34	34
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			27	27	27	27	27	27
StopCOM			28	28	28	28	28	28
ReadFlag			22	22	22	22	22	22
ResetFlag			29	29	29	29	29	29
ReceiveMessage	CCCA	14	117	117	117	223	223	223
	CCCB	15	223	223	223	223	223	223
GetMessageResource			69	69	69	69	69	69
ReleaseMessageResource			69	69	69	69	69	69
GetMessageStatus			85	85	85	85	85	85
SendMessage	SW CCCA	1, 14	139	139	139	251	251	251
	SW CCCB	1, 15	235	235	235	251	251	251
	NS CCCA	14	139	139	139	251	251	251
	NS CCCB	15	235	235	235	251	251	251
	KL CCCA	2, 14	110	110	110	224	224	224
	KL CCCB	2, 15	208	208	208	224	224	224
main_dispatch	NoHook	12	152	152	187	152	152	187

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
	Hook	13	176	176	219	176	176	219
sub_dispatch	B1LF	19	12	12	12	12	12	12
	B1HI	20	81	81	81	81	81	81
	B1HF	21	89	89	89	89	89	89
	B2LI	22	n/a	40	70	n/a	40	70
	B2LF	23	n/a	48	78	n/a	48	78
	B2HI	24	n/a	190	277	n/a	190	277
	B2HF	25	n/a	198	285	n/a	198	285
	E1HI	26	n/a	n/a	n/a	318	318	406
	E1HF	27	n/a	n/a	n/a	326	326	414
	E2HI	28	n/a	n/a	n/a	n/a	n/a	406
	E2HF	29	n/a	n/a	n/a	n/a	n/a	414
ErrorHook support		16	72	72	72	72	72	72
	ServiceID	17	78	78	78	78	78	78
	Parameters	18	99	99	99	99	99	99
validity_checks		3	36	36	36	36	36	36
Timing_dispatch		4	54	54	54	54	54	54
Timing_termination		4	108	108	108	108	108	108
ActivateTaskset	SW	1	245	289	340	256	306	370
	NS		297	340	391	308	357	421
	KL	2	191	234	285	202	252	314
ChainTaskset	SWL	1, 8	295	340	391	303	352	416
	SWH	1, 9	334	390	441	342	401	465
	NSL	8	365	410	461	373	422	486
	NSH	9	398	454	505	406	465	529
GetTasksetRef			104	104	104	104	104	104
MergeTaskset			201	201	201	201	201	201
AssignTaskset			155	155	155	155	155	155
RemoveTaskset			202	202	202	202	202	202
TestSubTaskset			216	216	216	216	216	216
TestEquivalentTaskset			214	214	214	214	214	214
TickSchedule	SW	1	281	249	249	249	249	249
	NS		343	339	339	339	339	339
	KL	2	230	202	202	202	202	202
AdvanceSchedule	SW	1	284	252	252	252	252	252
	NS		346	338	338	338	338	338
	KL	2	249	217	217	217	217	217

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			191	191	191	191	191	191
StopSchedule			137	137	137	137	137	137
GetScheduleStatus			177	177	177	177	177	177
GetScheduleValue			146	146	146	146	146	146
GetScheduleNext			84	84	84	84	84	84
SetScheduleNext			147	147	147	147	147	147
GetArrivalpointDelay			113	113	113	113	113	113
SetArrivalpointDelay			125	125	125	125	125	125
GetArrivalpointTasksetRef			109	109	109	109	109	109
GetArrivalpointNext			110	110	110	110	110	110
SetArrivalpointNext			171	171	171	171	171	171
TestArrivalpointWritable			124	124	124	124	124	124
GetExecutionTime			100	100	100	100	100	100
GetLargestExecutionTime			91	91	91	91	91	91
ResetLargestExecutionTime			85	85	85	85	85	85
GetStackOffset			18	18	18	18	18	18

Notes

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

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

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	122	162	197	125	162	214
	NS	106	148	178	110	147	199
	KL	63	102	136	65	99	151
TerminateTask	LExt	0	0	0	0	0	0
	H	197	199	197	198	196	197
ChainTask	SWL	269	319	387	337	372	457
	SWH	354	404	473	421	455	540
	NSL	269	319	390	337	372	454
	NSH	350	401	470	417	451	536
Schedule	SW	100	109	123	99	99	124
GetTaskID		53	44	45	53	53	45
GetTaskState		115	115	113	127	127	123
EnableAllInterrupts		39	39	41	40	40	42
DisableAllInterrupts		36	45	46	44	44	45
ResumeAllInterrupts		55	55	55	56	56	56
SuspendAllInterrupts		46	46	46	45	45	45
ResumeOSInterrupts		55	55	55	56	56	56
SuspendOSInterrupts		46	46	46	45	45	45
GetResource	Task	59	50	51	59	59	51
	Combined	105	105	105	105	105	105
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	104	104	101	101	101	104
	Combined	106	106	104	104	104	106
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	123	123	129
	NS	n/a	n/a	n/a	115	115	116
	KL	n/a	n/a	n/a	72	72	74
ClearEvent		n/a	n/a	n/a	59	59	60
GetEvent		n/a	n/a	n/a	44	44	43
WaitEvent	<default>	n/a	n/a	n/a	560	558	616
	fp	n/a	n/a	n/a	573	571	621
GetAlarmBase		122	122	121	115	115	123
GetAlarm		118	118	118	120	120	120

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		171	171	167	173	173	173
SetAbsAlarm		171	171	167	166	166	170
CancelAlarm		98	98	100	97	97	99
InitCounter		115	115	115	113	113	113
GetCounterValue		115	115	115	116	116	116
osek_tick_alarm	<default>	132	132	132	131	131	131
	KL	83	83	83	84	84	84
osek_incr_counter		12	12	12	12	12	12
GetActiveApplicationMode		7	7	7	15	15	15
StartOS		846	842	840	841	842	846
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	66	66	62	66	66	66
InitCOM		23	23	23	24	24	24
CloseCOM		15	15	15	14	14	14
StartCOM		50	50	51	156	154	154
StopCOM		29	29	29	29	29	29
ReadFlag		n/a	n/a	n/a	8	8	8
ResetFlag		n/a	n/a	n/a	8	8	8
ReceiveMessage		92	92	94	318	318	323
GetMessageResource		n/a	n/a	n/a	115	115	113
ReleaseMessageResource		n/a	n/a	n/a	153	153	158
GetMessageStatus		n/a	n/a	n/a	72	72	72
SendMessage	SW	225	265	302	453	490	539
	NS	209	260	283	429	466	524
	KL	123	162	196	341	375	428
ActivateTaskset	SW	117	448	490	114	449	509
	NS	93	433	476	97	435	486
	KL	49	388	430	52	387	436
	SW2	108	448	490	114	449	500
	NS2	93	433	476	97	435	486
	KL2	49	388	430	52	387	436
ChainTaskset	SWL	258	609	683	322	659	744
	SWH	345	694	770	408	745	829
	NSL	258	609	684	322	659	743
	NSH	341	690	768	404	741	829
GetTasksetRef		40	40	41	42	42	41

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		93	93	95	96	96	94
AssignTaskset		39	39	38	40	40	39
RemoveTaskset		95	95	96	97	97	96
TestSubTaskset		106	115	108	107	107	107
TestEquivalentTaskset		104	104	104	105	105	105
TickSchedule	SW	194	572	625	245	588	634
	NS	167	560	601	223	566	612
	KL	121	514	554	177	520	566
	SW2	185	572	616	236	571	620
	NS2	167	560	601	223	558	607
	KL2	121	523	554	177	512	561
AdvanceSchedule	SW	167	550	586	222	565	607
	NS	144	538	580	199	542	590
	KL	100	494	530	155	498	540
	SW2	158	550	586	213	548	593
	NS2	144	538	580	199	534	585
	KL2	100	494	530	155	490	535
StartSchedule		131	131	131	132	132	132
StopSchedule		104	104	105	104	104	104
GetScheduleStatus		118	118	117	119	119	118
GetScheduleValue		113	113	117	114	114	118
GetScheduleNext		41	41	42	42	42	43
SetScheduleNext		46	46	45	47	47	47
GetArrivalpointDelay		48	48	49	48	48	49
SetArrivalpointDelay		45	45	44	43	43	44
GetArrivalpointTasksetRef		36	36	37	35	35	36
GetArrivalpointNext		40	40	40	39	39	39
SetArrivalpointNext		39	39	39	38	38	38
TestArrivalpointWritable		52	52	51	51	51	51
GetExecutionTime		25	25	25	15	15	15
GetLargestExecutionTime		41	41	42	43	43	42
ResetLargestExecutionTime		23	23	23	23	23	23
GetStackOffset		50	51	51	51	51	51

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	121	162	192	125	162	209
	NS	107	148	182	110	147	194
	KL	61	102	134	67	99	151
TerminateTask	LExt	0	0	0	0	0	0
	H	468	470	470	467	470	470
ChainTask	SWL	585	628	702	652	681	770
	SWH	654	703	777	719	754	845
	NSL	585	628	699	652	681	773
	NSH	650	700	772	715	750	841
Schedule	SW	99	100	115	100	99	114
GetTaskID		53	45	45	53	45	45
GetTaskState		115	113	113	127	123	123
EnableAllInterrupts		41	39	39	42	40	40
DisableAllInterrupts		46	45	45	36	44	44
ResumeAllInterrupts		55	55	55	56	56	56
SuspendAllInterrupts		46	46	46	45	45	45
ResumeOSInterrupts		55	55	55	56	56	56
SuspendOSInterrupts		46	46	46	45	45	45
GetResource	Task	59	58	60	59	58	60
	Combined	105	105	105	105	105	105
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	104	101	101	101	104	104
	Combined	106	104	104	104	106	106
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	123	123	127
	NS	n/a	n/a	n/a	113	115	116
	KL	n/a	n/a	n/a	72	72	71
ClearEvent		n/a	n/a	n/a	60	59	59
GetEvent		n/a	n/a	n/a	43	44	44
WaitEvent	<default>	n/a	n/a	n/a	827	819	862
	fp	n/a	n/a	n/a	848	830	885
GetAlarmBase		112	113	113	114	115	115
GetAlarm		118	118	118	120	120	120

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		167	171	171	169	173	169
SetAbsAlarm		167	171	171	170	166	166
CancelAlarm		100	98	98	99	97	97
InitCounter		115	115	115	113	113	113
GetCounterValue		115	115	115	116	116	116
osek_tick_alarm	<default>	132	132	132	131	131	131
	KL	83	83	83	84	84	93
osek_incr_counter		12	12	12	12	12	12
GetActiveApplicationMode		7	7	7	6	6	6
StartOS		2551	2561	2557	2556	2557	2551
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	62	66	66	62	66	62
InitCOM		14	14	14	15	15	15
CloseCOM		15	15	15	14	14	14
StartCOM		51	50	50	155	156	157
StopCOM		29	29	29	29	29	29
ReadFlag		n/a	n/a	n/a	8	8	8
ResetFlag		n/a	n/a	n/a	8	8	8
ReceiveMessage		92	94	94	321	320	320
GetMessageResource		n/a	n/a	n/a	114	114	114
ReleaseMessageResource		n/a	n/a	n/a	153	158	158
GetMessageStatus		n/a	n/a	n/a	72	72	72
SendMessage	SW	226	265	295	450	481	528
	NS	212	251	285	435	466	513
	KL	121	162	194	344	375	427
ActivateTaskset	SW	111	448	488	111	449	496
	NS	94	433	473	96	435	481
	KL	49	388	428	52	387	440
	SW2	111	448	488	111	449	496
	NS2	94	433	473	96	435	481
	KL2	49	388	428	52	387	440
ChainTaskset	SWL	557	918	996	621	968	1059
	SWH	653	993	1075	714	1044	1138
	NSL	557	918	995	621	968	1069
	NSH	648	989	1069	709	1040	1118
GetTasksetRef		40	41	41	42	41	41

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		93	95	95	96	94	94
AssignTaskset		38	39	39	39	40	40
RemoveTaskset		95	96	96	97	96	96
TestSubTaskset		108	106	106	109	107	109
TestEquivalentTaskset		104	104	104	105	105	105
TickSchedule	SW	182	572	612	239	579	630
	NS	171	560	600	223	566	614
	KL	126	514	554	176	520	567
	SW2	182	572	612	239	571	627
	NS2	171	560	600	223	558	611
	KL2	126	514	554	176	512	564
AdvanceSchedule	SW	159	550	590	211	556	606
	NS	146	538	578	203	542	592
	KL	101	494	534	153	498	548
	SW2	159	550	590	211	548	603
	NS2	146	538	578	203	534	589
	KL2	101	494	534	153	490	545
StartSchedule		131	131	131	132	132	132
StopSchedule		105	104	104	105	104	105
GetScheduleStatus		118	117	117	119	118	118
GetScheduleValue		113	117	117	114	118	118
GetScheduleNext		41	42	42	42	43	43
SetScheduleNext		45	46	46	46	47	46
GetArrivalpointDelay		49	48	48	49	48	48
SetArrivalpointDelay		44	45	45	44	43	43
GetArrivalpointTasksetRef		37	36	36	36	35	35
GetArrivalpointNext		40	40	40	39	39	39
SetArrivalpointNext		39	39	39	38	38	38
TestArrivalpointWritable		51	52	52	50	51	50
GetExecutionTime		156	157	148	156	146	155
GetLargestExecutionTime		52	53	53	54	53	53
ResetLargestExecutionTime		45	44	44	44	45	45
GetStackOffset		51	52	52	50	52	52

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	461	495	536	466	505	558
	NS	503	541	572	506	537	591
	KL	385	423	455	390	422	473
TerminateTask	LExt	531	532	534	528	530	533
	H	596	592	596	593	595	597
ChainTask	SWL	1024	1066	1146	1090	1122	1216
	SWH	1089	1135	1217	1157	1195	1285
	NSL	1072	1122	1198	1139	1174	1264
	NSH	1138	1181	1262	1208	1240	1333
Schedule	SW	175	173	189	173	173	189
GetTaskID		59	49	50	49	49	50
GetTaskState		466	469	469	471	471	470
EnableAllInterrupts		47	45	47	55	46	48
DisableAllInterrupts		42	51	51	41	50	50
ResumeAllInterrupts		66	70	70	71	71	71
SuspendAllInterrupts		52	53	53	51	51	52
ResumeOSInterrupts		66	70	70	71	71	71
SuspendOSInterrupts		52	53	53	51	51	52
GetResource	Task	667	643	375	701	701	431
	Combined	360	355	359	413	413	416
	CLEx	386	382	385	440	440	442
ReleaseResource	Task	379	377	378	436	436	435
	Combined	356	357	357	415	415	414
	CLEx	364	366	366	421	421	420
SetEvent	SW	n/a	n/a	n/a	483	474	474
	NS	n/a	n/a	n/a	499	499	501
	KL	n/a	n/a	n/a	418	418	418
ClearEvent		n/a	n/a	n/a	109	109	109
GetEvent		n/a	n/a	n/a	366	366	369
WaitEvent	<default>	n/a	n/a	n/a	948	950	989
	fp	n/a	n/a	n/a	958	960	1000
GetAlarmBase		325	321	343	324	324	345
GetAlarm		333	330	351	333	333	353

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		441	443	463	443	443	464
SetAbsAlarm		416	415	435	415	415	433
CancelAlarm		315	314	334	315	315	333
InitCounter		461	464	485	463	463	483
GetCounterValue		307	299	307	301	301	307
osek_tick_alarm	<default>	172	170	170	171	171	169
	KL	83	83	83	84	84	84
osek_incr_counter		12	12	12	12	12	12
GetActiveApplicationMode		7	7	7	6	6	6
StartOS		2615	2610	2606	2615	2614	2609
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	64	67	67	64	64	67
InitCOM		14	14	14	15	15	15
CloseCOM		15	15	15	14	14	14
StartCOM		59	58	58	163	166	165
StopCOM		39	39	39	48	39	39
ReadFlag		n/a	n/a	n/a	44	44	44
ResetFlag		n/a	n/a	n/a	47	47	47
ReceiveMessage		260	256	256	498	498	497
GetMessageResource		n/a	n/a	n/a	589	589	589
ReleaseMessageResource		n/a	n/a	n/a	592	592	590
GetMessageStatus		n/a	n/a	n/a	185	185	187
SendMessage	SW	726	755	796	961	1000	1052
	NS	764	805	836	991	1022	1075
	KL	584	623	655	815	847	897
ActivateTaskset	SW	558	908	946	563	898	952
	NS	590	940	981	599	934	989
	KL	479	827	866	488	818	871
	SW2	558	908	946	563	898	952
	NS2	590	940	981	599	934	989
	KL2	479	827	866	488	818	871
ChainTaskset	SWL	1141	1501	1587	1209	1542	1642
	SWH	1213	1567	1659	1291	1615	1716
	NSL	1191	1552	1638	1260	1596	1691
	NSH	1264	1614	1704	1332	1668	1762
GetTasksetRef		347	349	349	351	351	350

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		234	232	232	233	233	233
AssignTaskset		143	143	142	143	143	143
RemoveTaskset		236	231	232	237	237	237
TestSubTaskset		247	252	253	248	248	253
TestEquivalentTaskset		244	250	251	245	245	251
TickSchedule	SW	295	1073	1113	735	1083	1132
	NS	345	1125	1164	787	1135	1184
	KL	213	997	1036	655	1003	1056
	SW2	295	1073	1113	735	1065	1117
	NS2	345	1125	1164	787	1117	1169
	KL2	213	997	1036	655	985	1041
AdvanceSchedule	SW	279	1059	1100	723	1071	1121
	NS	324	1109	1141	762	1110	1160
	KL	198	982	1021	642	990	1040
	SW2	279	1059	1100	723	1053	1106
	NS2	324	1109	1141	762	1092	1145
	KL2	198	982	1021	642	972	1025
StartSchedule		231	230	231	232	232	231
StopSchedule		179	180	181	179	179	180
GetScheduleStatus		194	200	195	201	201	195
GetScheduleValue		190	189	191	190	190	191
GetScheduleNext		73	76	73	77	77	74
SetScheduleNext		141	139	139	142	142	140
GetArrivalpointDelay		105	106	105	107	107	106
SetArrivalpointDelay		116	117	117	116	116	116
GetArrivalpointTasksetRef		82	81	82	80	80	81
GetArrivalpointNext		84	82	84	81	81	83
SetArrivalpointNext		158	160	160	159	159	159
TestArrivalpointWritable		95	98	98	94	94	97
GetExecutionTime		210	207	211	206	206	209
GetLargestExecutionTime		338	334	337	336	336	338
ResetLargestExecutionTime		330	326	326	327	327	326
GetStackOffset		51	51	52	50	50	51

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	22	22	22	22	22	22
	Cat 2	61	85	83	79	81	77

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	22	22	22	22	22	22
	Cat 2	239	259	253	253	247	247

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	22	22	22	22	22	22
	Cat 2	233	247	255	259	259	259

4.3.4 Task Switching Times

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

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

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

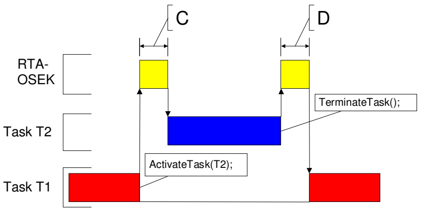

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

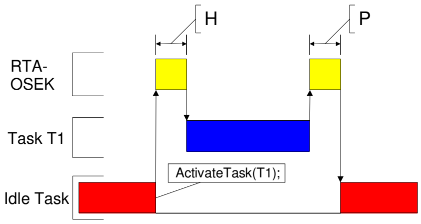

Figure 3: Task Activation from Idle Task

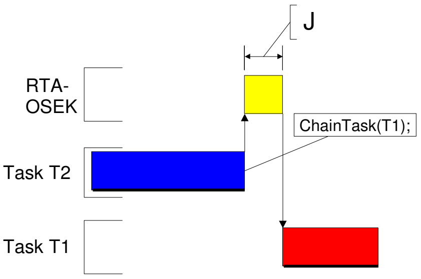

Figure 2: Task Chaining

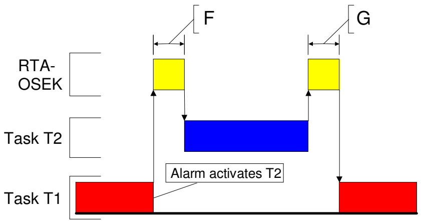

Figure 4: Task Activation from an Alarm

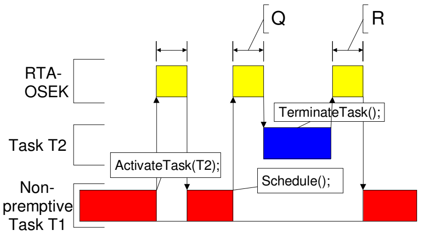

Figure 5: Non-Premptive Task Calls Schedule()

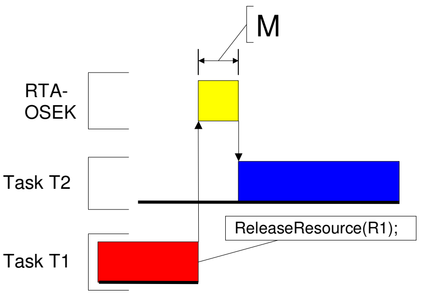

Figure 6: Blocked Task Activated by ReleaseResource()

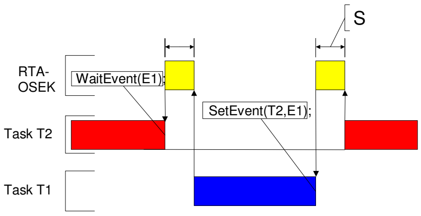

Figure 7: Waiting Task Activated by SetEvent()

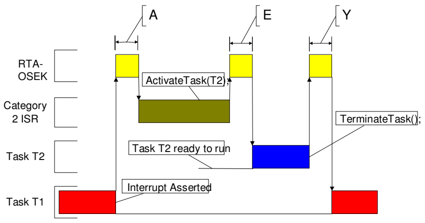

Figure 8: Category 2 ISR Activates a Higher Priority Task

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Normal termination	Light, Basic	98	146	167	100	143	171
Figure 1: D	Heavy, Basic/Extended	195	230	254	240	238	265
ChainTask	Light, Basic	188	245	313	191	244	327
Figure 2: J	Heavy, Basic/Extended	516	608	701	564	616	725
Pre-emption	Light, Basic	152	208	283	154	210	293
Figure 1: C	Heavy, Basic/Extended	239	288	363	305	345	432
From idle task	Light, Basic	152	208	283	154	210	293
Figure 3: H	Heavy, Basic/Extended	239	288	363	305	345	432
Triggered by alarm	Light, Basic	298	354	429	300	354	437
Figure 4: F	Heavy, Basic/Extended	383	432	507	449	491	578
Schedule	Light, Basic	123	139	194	118	138	194
Figure 5: Q	Heavy, Basic/Extended	210	219	274	269	273	326
Release resource	Light, Basic	140	156	197	138	158	196
Figure 6: M	Heavy, Basic/Extended	227	236	277	289	293	328
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	568	572	691
From category 2 ISR	Light, Basic	110	145	187	127	143	186
Figure 8: E	Heavy, Basic/Extended	197	225	267	278	278	318

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	379	411	434	380	411	439
Figure 1: D	Heavy, Basic/Extended	466	490	517	499	499	524
ChainTask	Light, Basic	513	555	630	515	555	645
Figure 2: J	Heavy, Basic/Extended	1108	1177	1281	1128	1176	1290
Pre-emption	Light, Basic	341	388	469	344	389	485
Figure 1: C	Heavy, Basic/Extended	419	467	547	486	522	615
From idle task	Light, Basic	341	388	469	344	389	485
Figure 3: H	Heavy, Basic/Extended	419	467	547	486	522	615
Triggered by alarm	Light, Basic	485	532	613	488	535	631
Figure 4: F	Heavy, Basic/Extended	565	613	693	632	666	759
Schedule	Light, Basic	309	319	385	312	317	383
Figure 5: Q	Heavy, Basic/Extended	387	398	463	454	450	514
Release resource	Light, Basic	329	336	387	329	337	386
Figure 6: M	Heavy, Basic/Extended	407	415	465	471	470	517
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	750	743	862
From category 2 ISR	Light, Basic	570	596	641	590	590	639
Figure 8: E	Heavy, Basic/Extended	648	675	719	732	723	770

Extended

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		40	40	40	40	40	40
BCC1 lightweight, floating-point		48	48	48	48	48	48
BCC1 heavyweight, integer		70	70	70	70	70	70
BCC1 heavyweight, floating-point		70	70	70	70	70	70
BCC2 lightweight, integer		n/a	48	48	n/a	48	48
BCC2 lightweight, floating-point		n/a	48	48	n/a	48	48
BCC2 heavyweight, integer		n/a	76	76	n/a	76	76
BCC2 heavyweight, floating-point		n/a	76	76	n/a	76	76
ECC1 heavyweight, integer		n/a	n/a	n/a	84	84	84
ECC1 heavyweight, floating-point		n/a	n/a	n/a	84	84	84
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	84
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	84
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		40	40	40	40	40	40
BCC1 lightweight, floating-point		48	48	48	48	48	48
BCC1 heavyweight, integer		70	70	70	70	70	70
BCC1 heavyweight, floating-point		70	70	70	70	70	70
BCC2 lightweight, integer		n/a	48	48	n/a	48	48
BCC2 lightweight, floating-point		n/a	48	48	n/a	48	48
BCC2 heavyweight, integer		n/a	76	76	n/a	76	76

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC2 heavyweight, floating-point		n/a	76	76	n/a	76	76
ECC1 heavyweight, integer		n/a	n/a	n/a	84	84	84
ECC1 heavyweight, floating-point		n/a	n/a	n/a	84	84	84
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	84
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	84

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		66	66	66	66	66	66
BCC1 lightweight, floating-point		70	70	70	70	70	70
BCC1 heavyweight, integer		96	96	96	96	96	96
BCC1 heavyweight, floating-point		96	96	96	96	96	96
BCC2 lightweight, integer		n/a	74	74	n/a	74	74
BCC2 lightweight, floating-point		n/a	74	74	n/a	74	74
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	118	118	118
ECC1 heavyweight, floating-point		n/a	n/a	n/a	118	118	118
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	118
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	118
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		64	64	66	64	64	66
BCC1 lightweight, floating-point		68	68	70	68	68	70
BCC1 heavyweight, integer		94	94	96	94	94	96
BCC1 heavyweight, floating-point		94	94	96	94	94	96
BCC2 lightweight, integer		n/a	72	74	n/a	72	74
BCC2 lightweight, floating-point		n/a	72	74	n/a	72	74

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC2 heavyweight, integer		n/a	104	106	n/a	104	106
BCC2 heavyweight, floating-point		n/a	104	106	n/a	104	106
ECC1 heavyweight, integer		n/a	n/a	n/a	116	116	118
ECC1 heavyweight, floating-point		n/a	n/a	n/a	116	116	118
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	118
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	118

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		66	66	66	66	66	66
BCC1 lightweight, floating-point		70	70	70	70	70	70
BCC1 heavyweight, integer		96	96	96	96	96	96
BCC1 heavyweight, floating-point		96	96	96	96	96	96
BCC2 lightweight, integer		n/a	74	74	n/a	74	74
BCC2 lightweight, floating-point		n/a	74	74	n/a	74	74
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	118	118	118
ECC1 heavyweight, floating-point		n/a	n/a	n/a	118	118	118
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	118
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	118
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		64	64	66	64	64	66
BCC1 lightweight, floating-point		68	68	70	68	68	70
BCC1 heavyweight, integer		94	94	96	94	94	96
BCC1 heavyweight, floating-point		94	94	96	94	94	96
BCC2 lightweight, integer		n/a	72	74	n/a	72	74

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	72	74	n/a	72	74
BCC2 heavyweight, integer		n/a	104	106	n/a	104	106
BCC2 heavyweight, floating-point		n/a	104	106	n/a	104	106
ECC1 heavyweight, integer		n/a	n/a	n/a	116	116	118
ECC1 heavyweight, floating-point		n/a	n/a	n/a	116	116	118
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	118
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	118

5 Compatibility with Pre-v5 Kernels

5.1 Updating the Application Version

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

5.2 Referencing the Stack section

The v3.20 and v5.00 kernels use the label `_systemstack_top` to reference the top of the system stack. This label is contained in the default Softune V30L33 startup routine contained in `start.asm`. Earlier versions of RTA-OSEK use a different label `SSTACK_TOP` that was needed to be added into `start.asm` by the user.

Important: Please ensure that label `_systemstack_top` marks the top of the system stack section.

5.3 32 Bit Timer Drivers

The v3.x kernels uses 16 bit timer values, whereas the v5.00 kernel uses 32 bit timer values. Therefore any existing applications' timer drivers will need modifying. Since the 16LX Timer Module provides only 16 bit timer registers the upper 16 bits will need to be emulated in software. The provided example application demonstrates one method of achieving this for the `TMRLR0` timer register.

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.