RTA-OSEK

Binding Manual: TMS470/TI

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: M00088-001

Disclaimer

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

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

Trademarks

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

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

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

Contents

1 About this Guide

This guide provides target-specific information for the TMS470/TI 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 Compiler

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

Vendor	Texas Instruments Inc.
Compiler	TMS470 C/C++ Compiler
Version	v4.4.1

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

Option	Description
`-c`	Compile only, do not link
`--abi=tiabi`	use TI ABI

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

Option	Description
`--abi=eabi`	use ARM ABI

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

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

Option	Description
`-c`	Compile only, do not link
`--abi=tiabi`	use TI ABI
`-mt/--code_state=16`	Generate 16-bit code
`--symdebug:none`	No debug output

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

Option	Description
`--abi=eabi`	use ARM ABI

2.1.1 Using the 16-Bit/32-Bit Assembler Instruction Set

The TI compiler can generate object code using either the TMS470 16-bit or 32-bit instruction set. The RTA-OSEK run-time libraries have been compiled using the 16-bit instruction set to reduce the amount of code memory used. The libraries support applications using either the 16-bit or the 32-bit instruction set and have been fully tested with both options.

All C files that contain tasks (i.e. functions decorated with the OSEK standard `TASK()` macro), Category 2 ISRs (i.e. functions decorated with the OSEK standard `ISR()` macro), function callbacks and hooks must be compiled using either the 16 bit instruction set or the 32 bit instruction set using inter-working support.

As the TMS470 CPU automatically switches to the 32-bit instruction set when an interrupt triggers Category 1 interrupt handlers require user generated veneers if compiled using the 16-bit instruction set, please refer to section 3.1.8 for further details.

Please consult the TI compiler documentation for a full explanation of 16/32-bit inter-working and how to employ it in your application.

2.2 Assembler

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

Vendor	Texas Instruments Inc.
Assembler	TMS470 Assembler
Version	v4.4.1

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

Option	Description
`-c`	Compile only, do not link
`--abi=tiabi`	use TI ABI

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

Option	Description
`--abi=eabi`	use ARM ABI

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
`-c`	Compile only, do not link
`--abi=tiabi`	use TI ABI

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

Option	Description
`--abi=eabi`	use ARM ABI

2.3 Linker/Locator

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

Option	Description
`--abi=tiabi`	use TI ABI

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

Option	Description
`--abi=eabi`	use ARM ABI

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 by RTA-OSEK GUI
`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
`IND$CALL`	Indirect function call

2.3.1 Stack Initialization for Different CPU Modes

The TMS470 CPU can use up to six different stacks, depending upon the number of operating modes that occur in an application.

RTA-OSEK uses two modes, supervisor (SVC) mode and IRQ mode (when Category 2 interrupts occur in an application). FIQ mode will only be used in applications that use Category 1 FIQ interrupts and, therefore, operate outside of the scope of RTA-OSEK.

All stack pointers must be initialized before use. It should be noted that it is your responsibility to ensure that each stack is large enough to avoid overrun.

Important: The initial SVC stack pointer (R13) value must be initialized to match the address of the symbol `_os_stack_top_svc`.

The example application demonstrates a suggested stack pointer initialization method. The start-up code, `init.asm`, declares the each stack section with the lengths defined by `STACK_LEN_<mode>` values. The size of the SVC stack can be defined using the `-stack` linker option. The linker command file, `linktms.cmd`, locates the stack's sections in memory and defines labels at the top of each stack section, `_os_stack_top_<mode>`.

2.3.2 Linker Command File

The RAM sections that are dedicated to RTA-OSEK consist of the `os_pur` and `os_pir` sections. These sections are initialized by the RTA-OSEK Component within the `StartOS()` API call (unlike standard C RAM variable sections, which are initialized in the application start-up code). These sections should be marked `NOLOAD` in the linker command file. This prevents the contents, relocation information and line number information being placed in the output module

The linker allocates space for the section and its symbols and it appears in the memory map listing. The example application contains an example of a linker command file.

2.4 Debugger

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

ORTI compatible debuggers	Lauterbach Trace32

The RTA-OSEK GUI outputs a file with the extension `.ort`. This file should be loaded into the debugger with the command `Task.ORTI <file>`. Note

that this must be loaded *after* the executable (`.out`) file. Please refer to the debugger documentation for further details on its support for ORTI.

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for TMS470/TI. 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 *TMS470R1x User's Guide*.

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

IPL Values	CPSR I-bit	CPSR F-bit	Description
0	0	0	User Level
N/A	0	1	Prohibited
1	1	0	OS Level - IRQ Category 1 and 2 interrupt level
2	1	1	Non IRQ Category 1 interrupt level

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	Description	Legality
0x04	Undefined instructions	Category 1
0x08	SWI	Category 1
0x0C	Prefetch Abort	Category 1
0x10	Data Abort	Category 1
0x18	IRQ (General Interrupt)	Single Category 1 or 2
0x1C	FIQ (Fast Interrupt)	Single Category 1
0x20	CIM channel 0	Category 1 or 2
0x24 to 0x98 in 4 byte steps	CIM channels 1 to 30	Category 1 or 2
0x9C	CIM channel 31	Category 1 or 2
0xA0	Phantom Interrupt	Category 1 or 2

The valid base addresses for the vector table are:

Base Address	Notes
0x0	ARM CPU vector table

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

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

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

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

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

Vector Location	Label
`0xVV`	`_os_wrapper_VV`
`e.g. 0x24`	`_os_wrapper_24`

3.1.6 Software Vector Table

The vector table used in the TMS470R1 implementation of RTA-OSEK splits the ARM CPU vector table and the 32 interrupt channels of the Central Interrupt Manager (CIM) (see Section 3.1.2). The 32 CIM channels are characterized in an integer array `os_irq_fiq_vectors`, which contains the addresses of the interrupt handlers for ISRs on these vectors. The offset addresses shown in Section 3.1.2 illustrates the direct mapping between each array entry and the interrupt source that should be used in the OIL configuration file.

3.1.7 Reset Vector

In the generated vector table, the Reset Vector at 0x0 is always attached to a user function `c_int00()`. This reset handler function should perform the start-up operation required for the TI C compiler and set-up the stack pointers for the CPU modes used. An example of `c_int00()` is provided in the example application.

3.1.8 Handling Multiple Interrupt Sources

The CIM allows multiple interrupt sources to be processed and handled by the CPU FIQ and IRQ interrupt lines. RTA-OSEK supports this using the `os_irq_entry()` and `os_fiq_entry()` functions. These functions are the multiple interrupt handler entry points. They reference the CIM hardware registers to determine the interrupt source.

If RTA-OSEK generates the vector table and multiple FIQ and IRQ interrupts are contained within the application OIL configuration file, these functions are automatically located on vector table addresses 0x18 and 0x1C. If a user generated vector table is provided for an application with multiple interrupts bound on FIQ/IRQ, the appropriate functions should be bound to addresses 0x18 and 0x1C.

When multiple interrupts occur within the application, the entry functions of all Category 1 interrupts should be standard C functions. They should not be decorated with the `interrupt` modifier. Category 2 interrupts should still be decorated with the OSEK standard `ISR()` macro. Extra context handling required by interrupt functions is dealt with by `os_irq_entry()` and `os_fiq_entry()`.

3.1.9 Handling single IRQ/FIQ interrupt sources

The CIM does not need to be referenced to determine the interrupt source in applications where single interrupts trigger the IRQ or FIQ interrupt line. Instead, the interrupt can be bound directly to the CPU IRQ (0x18) or FIQ (0x1C) vector, consequently reducing the interrupt entry time. Single

Category 2 IRQ interrupts should be bound to vector 0x18 in the OIL configuration file

The entry function that occurs on the wrapper is `_os_wrapper_018()`. If a user generated vector table is used with a single Category 2 IRQ interrupt, the assembler in Code Example 3:2 can be used to implement this vector table.

```
  .sect ".os_intvec"
  .align 4
  LDR   pc,_os_reset_addr
  B     $
  B     $
  B     $
  B     $
  NOP
  LDR   pc,_os_irq_addr
  B     $
  .ref  _c_int00
_os_reset_addr:   .long _c_int00
_os_irq_addr:     .long _os_wrapper_018
```

Code Example 3:2 - Implementing a Vector Table with a Single Category 2 ISR

3.1.10 Phantom interrupt support

A phantom interrupt occurs when an interrupt is triggered and the CPU cannot identify the source. This occurs when the IRQ or FIQ request signal is high but the contents of the CIM interrupt offset register (IRQIVEC/FIQIVEC) and interrupt request register (INTREQ) are zero.

RTA-OSEK supports a phantom interrupt handler to be attached to a VIM offset of 0. This can be entered as a result of an FIQ or IRQ interrupt. The IPL of the phantom interrupt should be either 1 for IRQ interrupts or 2 for FIQ interrupts.

3.2 CPU Operating Modes

All tasks and Category 2 ISRs execute in supervisor mode. When a Category 2 interrupt occurs, the RTA-OSEK Component switches from IRQ mode to supervisor (SVC) mode before processing the interrupt. This minimizes the worst-case stack requirements.

From reset, the processor runs on the SVC stack. The RTA-OSEK Component always expects to run on the SVC stack (i.e. in SVC mode), except when processing Category 1 interrupts. Category 1 interrupts can be configured to use the following CPU operating modes; SVC by using the SWI, FIQ, IRQ, Abort and Undefined.

3.3 Register Settings

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

Register	Required Value
FIRQPR	Initialize with OS_FIRQPR_INIT

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

Register	Notes
CPSR (CPU)	Used to control the I and F bits and CPU mode

3.4 Stack Usage

3.4.1 Number of Stacks

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

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

3.4.2 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): 24

Timing

API max usage (bytes): 24

Extended

API max usage (bytes): 24

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.

3.4.3 Stack Usage with 16-Bit/32-Bit Instructions

The RTA-OSEK runtime libraries can support applications written in both the 16-bit and 32-bit TMS470 instruction sets. The libraries were compiled using the 16-bit instruction set. Stack usage will change when applications use 32-bit instructions.

The 16-bit instruction set should be used to optimize stack usage. When the 32-bit instruction set is used, veneers are placed between functions compiled using the 16 and 32-bit instruction sets. Each veneer uses 4 bytes of stack, which should be taken into account during SVC mode stack calculations.

When an ECC task occurs in an application the amount of stack used by the task must be entered into the OIL configuration file. If the task is compiled using 32-bit instructions, the stack figure in the OIL file is the amount of stack used by the task with 8 bytes added (4 bytes for the veneer to get into the task and 4 bytes for the veneer to enter the `WaitEvent()` API call).

3.4.4 Stack Usage in Tick_<Counter Name> ()

In applications that use alarms, a function `Tick_<counter name>` (where `<counter name>` is the associated counter that the alarm is attached to) is placed in `osekdefs.c`.

The amount of stack used in this function depends upon the level of compiler optimization and the number of alarms implemented. A nominal stack usage of 16 bytes has been attributed to this function, which is sufficient for up to 8 alarms. If more alarms are used within an application that uses ECC tasks, then the number of stack bytes used by `Tick_<counter name>()` should be reviewed. Any extra stack usage should be added to the idle task stack figure in the OIL configuration file.

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	**TMS470R1**
Clock speed (MHz)	15
Code memory	Off-chip RAM
Read-only data memory	Off-chip RAM
Read-write data memory	On-chip RAM

4.1 Functionality

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

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

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	4294967295					

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	46	46	46	46	46	46
	ROM	160	160	168	206	206	214
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	66	66	66	66	66	66
	ROM	232	232	240	278	278	286
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	84	84	84	84	84	84
	ROM	284	284	292	328	328	336
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

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	36	36	36	36	36	36
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	40	40	40	40	40	40
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	48	56	n/a	48	56
ECC1, Integer task	RAM	n/a	n/a	n/a	56	56	56
	ROM	n/a	n/a	n/a	60	60	60
ECC1, floating-point task	RAM	n/a	n/a	n/a	58	58	58
	ROM	n/a	n/a	n/a	60	60	60
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	58
	ROM	n/a	n/a	n/a	n/a	n/a	68
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	60
	ROM	n/a	n/a	n/a	n/a	n/a	68
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	148	148	148	148	148	148
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	164	164	164	164	164	164
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	34	34	34	34	34	34
Counter	RAM	4	4	4	4	4	4
	ROM	64	66	66	66	66	66
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20

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	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	68	68	68	68	68	68
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	52	52	52	52	52	52
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	60	68	n/a	60	68
ECC1, Integer task	RAM	n/a	n/a	n/a	68	68	68
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating-point task	RAM	n/a	n/a	n/a	70	70	70
	ROM	n/a	n/a	n/a	72	72	72

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	70
	ROM	n/a	n/a	n/a	n/a	n/a	80
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	72
	ROM	n/a	n/a	n/a	n/a	n/a	80
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	174	174	174	174	174	174
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	182	182	182	182	182	182
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	34	34	34	34	34	34
Counter	RAM	4	4	4	4	4	4
	ROM	66	66	66	66	66	66
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	68	68	68	68	68	68
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	16	16	16	16	16	16
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	68	76	n/a	68	76
ECC1, Integer task	RAM	n/a	n/a	n/a	72	72	72
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	74	74	74
	ROM	n/a	n/a	n/a	80	80	80
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	74
	ROM	n/a	n/a	n/a	n/a	n/a	88
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	76
	ROM	n/a	n/a	n/a	n/a	n/a	88
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	186	186	186	186	186	186
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	194	194	194	194	194	194
Resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
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	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	38	38	38	38	38	38
Counter	RAM	4	4	4	4	4	4
	ROM	70	70	70	70	70	70
Message	RAM	11	11	11	31	31	31
	ROM	24	24	24	60	60	60
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	68	68	68	68	68	68
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Arrivalpoint (writable)	RAM	20	20	20	20	20	20
	ROM	20	20	20	20	20	20
Schedule	RAM	20	20	20	20	20	20
	ROM	44	44	44	44	44	44
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

4.2.3 Size of Linkable Modules

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

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

The call variants are as follows:

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

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	130	170	202	140	180	232
	NS		108	148	180	118	158	208
	KL	2	86	124	158	96	134	184
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	46	46	46	46	46	46
ChainTask	SWL	1, 8	114	152	182	124	162	210
	SWH	1, 9	142	182	212	152	192	244
	NSL	8	114	152	182	124	162	210
	NSH	9	130	170	200	140	180	232
Schedule			90	90	108	90	90	108
GetTaskID			40	40	40	40	40	40
GetTaskState			82	82	82	98	98	98
EnableAllInterrupts			40	40	40	40	40	40
DisableAllInterrupts			46	46	46	46	46	46
ResumeAllInterrupts			54	54	54	54	54	54
SuspendAllInterrupts			64	64	64	64	64	64
ResumeOSInterrupts			54	54	54	54	54	54
SuspendOSInterrupts			74	74	74	74	74	74
GetResource	Task	7	50	50	54	50	50	54
	Combined	6	70	70	70	70	70	70
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	76	76	76	76	76	76
	Combined	6	118	118	118	118	118	118
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	116	116	186
	NS		n/a	n/a	n/a	90	90	160
	NS1i	10	n/a	n/a	n/a	56	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	78	78	148
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	56	56	56
GetEvent			n/a	n/a	n/a	22	22	22
WaitEvent	<default>		n/a	n/a	n/a	184	184	314
	fp	11	n/a	n/a	n/a	206	206	364
	1i	10	n/a	n/a	n/a	30	n/a	n/a
GetAlarmBase			50	50	50	50	50	50
GetAlarm			88	88	88	88	88	88
SetRelAlarm			362	362	362	362	362	362
SetAbsAlarm			400	400	400	400	400	400
CancelAlarm			76	76	76	76	76	76
InitCounter			56	56	56	56	56	56
GetCounterValue			68	68	68	68	68	68
GetScheduleTableStatus		34	48	62	58	44	58	58
NextScheduleTable		34	62	154	150	58	150	150
StartScheduleTable		34	80	108	94	80	96	96
StopScheduleTable		34	66	86	78	64	78	78
ScheduleTable expiry point	ActivateTask		10	10	10	10	10	10
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	12	12	12
ScheduleTable expiry point	Callback		8	8	8	8	8	8
ScheduleTable expiry point	Tick counter		10	10	10	10	10	10
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	28	28	28	28	28	28
Process container	Non-Yielding	33	1192	1312	1380	1408	1520	1588
osek_tick_alarm	<default>		70	70	70	70	70	70
	KL	2	46	46	46	46	46	46
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			134	134	134	134	134	134
ShutdownOS	NoHook	12	42	42	42	42	42	42
	Hook	13	50	50	50	50	50	50
InitCOM			16	16	16	16	16	16
CloseCOM			16	16	16	16	16	16
StartCOM			42	42	42	42	42	42
StopCOM			32	32	32	32	32	32

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	60	60	60	152	152	152
	CCCB	15	152	152	152	152	152	152
GetMessageResource			60	60	60	60	60	60
ReleaseMessageResource			58	58	58	58	58	58
GetMessageStatus			50	50	50	50	50	50
SendMessage	SW CCCA	1, 14	80	80	80	182	182	182
	SW CCCB	1, 15	170	170	170	182	182	182
	NS CCCA	14	80	80	80	182	182	182
	NS CCCB	15	170	170	170	182	182	182
	KL CCCA	2, 14	64	64	64	168	168	168
	KL CCCB	2, 15	156	156	156	168	168	168
main_dispatch	NoHook	12	116	116	146	116	116	146
	Hook	13	152	152	182	152	152	182
sub_dispatch	B1LF	19	50	50	50	50	50	50
	B1HI	20	92	92	92	92	92	92
	B1HF	21	100	100	100	100	100	100
	B2LI	22	n/a	80	98	n/a	80	98
	B2LF	23	n/a	88	106	n/a	88	106
	B2HI	24	n/a	182	232	n/a	182	232
	B2HF	25	n/a	190	240	n/a	190	240
	E1HI	26	n/a	n/a	n/a	244	244	292
	E1HF	27	n/a	n/a	n/a	252	252	300
	E2HI	28	n/a	n/a	n/a	n/a	n/a	292
	E2HF	29	n/a	n/a	n/a	n/a	n/a	300
ErrorHook support		16	56	56	56	56	56	56
	ServiceID	17	64	64	64	64	64	64
	Parameters	18	76	76	76	76	76	76
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	126	232	270	140	258	328
	NS		104	208	246	118	234	302
	KL	2	82	190	222	96	216	288
ChainTaskset	SWL	1, 8	114	226	260	122	254	316
	SWH	1, 9	148	270	304	156	296	356

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	NSL	8	114	226	260	122	254	316
	NSH	9	136	258	292	144	286	344
GetTasksetRef			20	20	20	20	20	20
MergeTaskset			50	50	50	50	50	50
AssignTaskset			20	20	20	20	20	20
RemoveTaskset			50	50	50	50	50	50
TestSubTaskset			60	60	60	60	60	60
TestEquivalentTaskset			58	58	58	58	58	58
TickSchedule	SW	1	194	168	168	168	168	168
	NS		166	136	136	136	136	136
	KL	2	154	120	120	120	120	120
AdvanceSchedule	SW	1	170	154	154	154	154	154
	NS		144	122	122	122	122	122
	KL	2	132	110	110	110	110	110
StartSchedule			78	78	78	78	78	78
StopSchedule			62	62	62	62	62	62
GetScheduleStatus			90	90	90	90	90	90
GetScheduleValue			68	68	68	68	68	68
GetScheduleNext			22	22	22	22	22	22
SetScheduleNext			20	20	20	20	20	20
GetArrivalpointDelay			20	20	20	20	20	20
SetArrivalpointDelay			18	18	18	18	18	18
GetArrivalpointTasksetRef			18	18	18	18	18	18
GetArrivalpointNext			20	20	20	20	20	20
SetArrivalpointNext			18	18	18	18	18	18
TestArrivalpointWritable			42	42	42	42	42	42
GetExecutionTime			16	16	16	16	16	16
GetLargestExecutionTime			18	18	18	18	18	18
ResetLargestExecutionTime			16	16	16	16	16	16
GetStackOffset			n/a	n/a	n/a	n/a	n/a	n/a

Timing

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	130	170	202	140	180	232
	NS		108	148	180	118	158	208
	KL	2	86	124	158	96	134	184
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	46	46	46	46	46	46
ChainTask	SWL	1, 8	114	152	182	124	162	210
	SWH	1, 9	142	182	212	152	192	244
	NSL	8	114	152	182	124	162	210
	NSH	9	130	170	200	140	180	232
Schedule			110	110	128	110	110	128
GetTaskID			40	40	40	40	40	40
GetTaskState			82	82	82	98	98	98
EnableAllInterrupts			40	40	40	40	40	40
DisableAllInterrupts			46	46	46	46	46	46
ResumeAllInterrupts			54	54	54	54	54	54
SuspendAllInterrupts			64	64	64	64	64	64
ResumeOSInterrupts			54	54	54	54	54	54
SuspendOSInterrupts			74	74	74	74	74	74
GetResource	Task	7	50	50	54	50	50	54
	Combined	6	70	70	70	70	70	70
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	96	96	96	96	96	96
	Combined	6	154	154	154	154	154	154
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	116	116	186
	NS		n/a	n/a	n/a	90	90	160
	NS1i	10	n/a	n/a	n/a	56	n/a	n/a
	KL	2	n/a	n/a	n/a	78	78	148
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	56	56	56
GetEvent			n/a	n/a	n/a	22	22	22
WaitEvent	<default>		n/a	n/a	n/a	230	230	360
	fp	11	n/a	n/a	n/a	252	252	410

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	88	n/a	n/a
GetAlarmBase			50	50	50	50	50	50
GetAlarm			88	88	88	88	88	88
SetRelAlarm			362	362	362	362	362	362
SetAbsAlarm			400	400	400	400	400	400
CancelAlarm			76	76	76	76	76	76
InitCounter			56	56	56	56	56	56
GetCounterValue			68	68	68	68	68	68
GetScheduleTableStatus		34	44	58	58	44	58	58
NextScheduleTable		34	58	150	150	58	150	150
StartScheduleTable		34	78	94	94	80	96	96
StopScheduleTable		34	64	78	78	64	78	78
ScheduleTable expiry point	ActivateTask		10	10	10	10	10	10
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	12	12	12
ScheduleTable expiry point	Callback		8	8	8	8	8	8
ScheduleTable expiry point	Tick counter		10	10	10	10	10	10
ScheduleTable expiry point	Final		26	26	26	26	26	26
GetISRID		4	24	24	24	24	24	24
Process container	Yielding	32	28	28	28	28	28	28
Process container	Non-Yielding	33	1192	1312	1380	1408	1520	1588
osek_tick_alarm	<default>		70	70	70	70	70	70
	KL	2	46	46	46	46	46	46
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			180	180	180	180	180	180
ShutdownOS	NoHook	12	42	42	42	42	42	42
	Hook	13	50	50	50	50	50	50
InitCOM			16	16	16	16	16	16
CloseCOM			16	16	16	16	16	16
StartCOM			42	42	42	42	42	42
StopCOM			32	32	32	32	32	32
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	60	60	60	152	152	152
	CCCB	15	152	152	152	152	152	152
GetMessageResource			60	60	60	60	60	60
ReleaseMessageResource			58	58	58	58	58	58

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			50	50	50	50	50	50
SendMessage	SW CCCA	1, 14	80	80	80	182	182	182
	SW CCCB	1, 15	170	170	170	182	182	182
	NS CCCA	14	80	80	80	182	182	182
	NS CCCB	15	170	170	170	182	182	182
	KL CCCA	2, 14	64	64	64	168	168	168
	KL CCCB	2, 15	156	156	156	168	168	168
main_dispatch	NoHook	12	176	176	208	176	176	208
	Hook	13	214	214	246	214	214	246
sub_dispatch	B1LF	19	40	40	40	40	40	40
	B1HI	20	102	102	102	102	102	102
	B1HF	21	110	110	110	110	110	110
	B2LI	22	n/a	72	90	n/a	72	90
	B2LF	23	n/a	80	98	n/a	80	98
	B2HI	24	n/a	186	236	n/a	186	236
	B2HF	25	n/a	194	244	n/a	194	244
	E1HI	26	n/a	n/a	n/a	280	280	324
	E1HF	27	n/a	n/a	n/a	288	288	332
	E2HI	28	n/a	n/a	n/a	n/a	n/a	324
	E2HF	29	n/a	n/a	n/a	n/a	n/a	332
ErrorHook support		16	56	56	56	56	56	56
	ServiceID	17	64	64	64	64	64	64
	Parameters	18	76	76	76	76	76	76
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	82	82	82	82	82	82
Timing_termination		4	84	84	84	84	84	84
ActivateTaskset	SW	1	126	232	270	140	258	328
	NS		104	208	246	118	234	302
	KL	2	82	190	222	96	216	288
ChainTaskset	SWL	1, 8	114	226	260	122	254	316
	SWH	1, 9	148	270	304	156	296	356
	NSL	8	114	226	260	122	254	316
	NSH	9	136	258	292	144	286	344
GetTasksetRef			20	20	20	20	20	20
MergeTaskset			50	50	50	50	50	50
AssignTaskset			20	20	20	20	20	20
RemoveTaskset			50	50	50	50	50	50

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			60	60	60	60	60	60
TestEquivalentTaskset			58	58	58	58	58	58
TickSchedule	SW	1	194	168	168	168	168	168
	NS		166	136	136	136	136	136
	KL	2	154	120	120	120	120	120
AdvanceSchedule	SW	1	170	154	154	154	154	154
	NS		144	122	122	122	122	122
	KL	2	132	110	110	110	110	110
StartSchedule			78	78	78	78	78	78
StopSchedule			62	62	62	62	62	62
GetScheduleStatus			90	90	90	90	90	90
GetScheduleValue			68	68	68	68	68	68
GetScheduleNext			22	22	22	22	22	22
SetScheduleNext			20	20	20	20	20	20
GetArrivalpointDelay			20	20	20	20	20	20
SetArrivalpointDelay			18	18	18	18	18	18
GetArrivalpointTasksetRef			18	18	18	18	18	18
GetArrivalpointNext			20	20	20	20	20	20
SetArrivalpointNext			18	18	18	18	18	18
TestArrivalpointWritable			42	42	42	42	42	42
GetExecutionTime			102	102	102	102	102	102
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			24	24	24	24	24	24
GetStackOffset			n/a	n/a	n/a	n/a	n/a	n/a

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	186	228	258	200	238	290
	NS		220	262	292	234	272	324
	KL	2	142	182	212	156	192	244
TerminateTask	LExt	3	126	126	126	126	126	126

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	146	146	146	146	146	146
ChainTask	SWL	1, 8	226	266	294	238	276	324
	SWH	1, 9	256	300	328	270	310	360
	NSL	8	270	310	338	282	320	368
	NSH	9	292	334	362	304	344	394
Schedule			192	192	210	192	192	210
GetTaskID			54	54	54	54	54	54
GetTaskState			194	194	194	200	200	200
EnableAllInterrupts			54	54	54	54	54	54
DisableAllInterrupts			60	60	60	60	60	60
ResumeAllInterrupts			92	92	92	92	92	92
SuspendAllInterrupts			78	78	78	78	78	78
ResumeOSInterrupts			92	92	92	92	92	92
SuspendOSInterrupts			88	88	88	88	88	88
GetResource	Task	7	256	256	232	256	256	232
	Combined	6	236	236	236	236	236	236
	CLEx	3	224	224	224	224	224	224
ReleaseResource	Task	7	248	248	248	248	248	248
	Combined	6	300	300	300	300	300	300
	CLEx	3	204	204	204	204	204	204
SetEvent	SW	1	n/a	n/a	n/a	234	234	304
	NS		n/a	n/a	n/a	262	262	332
	NS1i	10	n/a	n/a	n/a	166	n/a	n/a
	KL	2	n/a	n/a	n/a	192	192	262
	KL1i	2, 10	n/a	n/a	n/a	142	n/a	n/a
ClearEvent			n/a	n/a	n/a	114	114	114
GetEvent			n/a	n/a	n/a	150	150	150
WaitEvent	<default>		n/a	n/a	n/a	314	314	440
	fp	11	n/a	n/a	n/a	336	336	490
	1i	10	n/a	n/a	n/a	178	n/a	n/a
GetAlarmBase			128	128	128	128	128	128
GetAlarm			138	138	138	138	138	138
SetRelAlarm			456	456	456	456	456	456
SetAbsAlarm			476	476	476	476	476	476
CancelAlarm			124	124	124	124	124	124
InitCounter			174	174	174	174	174	174
GetCounterValue			146	146	146	146	146	146

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	58	72	72	58	72	72
NextScheduleTable		34	68	160	160	68	160	160
StartScheduleTable		34	88	104	104	90	106	106
StopScheduleTable		34	74	88	88	74	88	88
ScheduleTable expiry point	ActivateTask		10	10	10	10	10	10
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	12	12	12
ScheduleTable expiry point	Callback		8	8	8	8	8	8
ScheduleTable expiry point	Tick counter		10	10	10	10	10	10
ScheduleTable expiry point	Final		26	26	26	26	26	26
GetISRID		4	34	34	34	34	34	34
Process container	Yielding	32	28	28	28	28	28	28
Process container	Non-Yielding	33	1444	1556	1628	1668	1772	1844
osek_tick_alarm	<default>		88	88	88	88	88	88
	KL	2	46	46	46	46	46	46
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			194	194	194	194	194	194
ShutdownOS	NoHook	12	52	52	52	52	52	52
	Hook	13	60	60	60	60	60	60
InitCOM			16	16	16	16	16	16
CloseCOM			16	16	16	16	16	16
StartCOM			56	56	56	56	56	56
StopCOM			54	54	54	54	54	54
ReadFlag			38	38	38	38	38	38
ResetFlag			40	40	40	40	40	40
ReceiveMessage	CCCA	14	128	128	128	218	218	218
	CCCB	15	218	218	218	218	218	218
GetMessageResource			96	96	96	96	96	96
ReleaseMessageResource			96	96	96	96	96	96
GetMessageStatus			98	98	98	98	98	98
SendMessage	SW CCCA	1, 14	152	152	152	252	252	252
	SW CCCB	1, 15	240	240	240	252	252	252
	NS CCCA	14	152	152	152	252	252	252
	NS CCCB	15	240	240	240	252	252	252
	KL CCCA	2, 14	134	134	134	234	234	234
	KL CCCB	2, 15	222	222	222	234	234	234
main_dispatch	NoHook	12	176	176	208	176	176	208

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	214	214	246	214	214	246
sub_dispatch	B1LF	19	40	40	40	40	40	40
	B1HI	20	102	102	102	102	102	102
	B1HF	21	110	110	110	110	110	110
	B2LI	22	n/a	72	90	n/a	72	90
	B2LF	23	n/a	80	98	n/a	80	98
	B2HI	24	n/a	186	236	n/a	186	236
	B2HF	25	n/a	194	244	n/a	194	244
	E1HI	26	n/a	n/a	n/a	280	280	324
	E1HF	27	n/a	n/a	n/a	288	288	332
	E2HI	28	n/a	n/a	n/a	n/a	n/a	324
	E2HF	29	n/a	n/a	n/a	n/a	n/a	332
ErrorHook support		16	106	106	106	106	106	106
	ServiceID	17	118	118	118	118	118	118
	Parameters	18	140	140	140	140	140	140
validity_checks		3	38	38	38	38	38	38
Timing_dispatch		4	82	82	82	82	82	82
Timing_termination		4	84	84	84	84	84	84
ActivateTaskset	SW	1	246	306	334	268	338	384
	NS		276	332	366	296	358	414
	KL	2	198	248	284	220	274	332
ChainTaskset	SWL	1, 8	290	362	392	310	388	434
	SWH	1, 9	336	408	442	358	436	484
	NSL	8	336	412	448	362	442	490
	NSH	9	370	450	484	396	480	526
GetTasksetRef			118	118	118	118	118	118
MergeTaskset			184	184	184	184	184	184
AssignTaskset			140	140	140	140	140	140
RemoveTaskset			184	184	184	184	184	184
TestSubTaskset			194	194	194	194	194	194
TestEquivalentTaskset			192	192	192	192	192	192
TickSchedule	SW	1	296	250	250	250	250	250
	NS		320	292	292	292	292	292
	KL	2	244	204	204	204	204	204
AdvanceSchedule	SW	1	274	254	254	254	254	254
	NS		318	292	292	292	292	292
	KL	2	236	210	210	210	210	210

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			178	178	178	178	178	178
StopSchedule			130	130	130	130	130	130
GetScheduleStatus			168	168	168	168	168	168
GetScheduleValue			142	142	142	142	142	142
GetScheduleNext			88	88	88	88	88	88
SetScheduleNext			150	150	150	150	150	150
GetArrivalpointDelay			114	114	114	114	114	114
SetArrivalpointDelay			128	128	128	128	128	128
GetArrivalpointTasksetRef			112	112	112	112	112	112
GetArrivalpointNext			114	114	114	114	114	114
SetArrivalpointNext			148	148	148	148	148	148
TestArrivalpointWritable			124	124	124	124	124	124
GetExecutionTime			130	130	130	130	130	130
GetLargestExecutionTime			106	106	106	106	106	106
ResetLargestExecutionTime			102	102	102	102	102	102
GetStackOffset			n/a	n/a	n/a	n/a	n/a	n/a

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 TMS470/TI port of the RTA-OSEK Component was achieved using a timer running two times slower than the CPU clock speed. The figures in this section, therefore, have an uncertainty level of up to two CPU cycles. The actual times are between 0 and two 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	99	131	171	107	121	177
	NS	85	119	157	93	109	161
	KL	51	85	123	57	75	125
TerminateTask	LExt	0	0	0	0	0	0
	H	135	135	141	135	135	143
ChainTask	SWL	257	303	367	313	329	405
	SWH	313	359	423	367	381	463
	NSL	257	303	369	311	329	405
	NSH	305	351	415	361	373	457
Schedule	SW	79	79	91	79	81	91
GetTaskID		29	29	29	29	29	29
GetTaskState		85	87	87	95	93	95
EnableAllInterrupts		31	31	31	31	31	31
DisableAllInterrupts		45	47	47	45	47	47
ResumeAllInterrupts		41	41	41	41	41	41
SuspendAllInterrupts		63	61	61	63	61	61
ResumeOSInterrupts		41	41	41	41	41	41
SuspendOSInterrupts		61	63	63	61	63	63
GetResource	Task	39	39	41	41	41	43
	Combined	55	55	55	57	57	57
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	75	75	75	77	77	77
	Combined	105	105	105	105	105	105
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	105	105	105
	NS	n/a	n/a	n/a	103	103	103
	KL	n/a	n/a	n/a	71	71	71
ClearEvent		n/a	n/a	n/a	63	63	63
GetEvent		n/a	n/a	n/a	23	23	23
WaitEvent	<default>	n/a	n/a	n/a	363	363	417

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	371	371	427
GetAlarmBase		69	69	69	67	67	69
GetAlarm		91	91	91	91	91	91
SetRelAlarm		125	125	125	125	125	125
SetAbsAlarm		107	107	107	107	107	107
CancelAlarm		71	71	71	73	73	71
InitCounter		65	65	65	65	65	65
GetCounterValue		71	71	71	69	69	71
osek_tick_alarm	<default>	79	81	81	79	81	79
	KL	41	41	41	39	39	41
osek_incr_counter		15	15	15	15	15	15
GetActiveApplicationMode		11	11	11	11	11	11
StartOS		1053	1053	1053	1053	1053	1053
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	49	49	49	49	49	49
InitCOM		7	7	9	7	9	7
CloseCOM		9	7	9	9	9	7
StartCOM		31	31	31	101	99	101
StopCOM		17	17	17	17	17	17
ReadFlag		n/a	n/a	n/a	17	19	17
ResetFlag		n/a	n/a	n/a	13	13	13
ReceiveMessage		67	67	67	253	253	253
GetMessageResource		n/a	n/a	n/a	89	89	89
ReleaseMessageResource		n/a	n/a	n/a	123	123	123
GetMessageStatus		n/a	n/a	n/a	39	39	39
SendMessage	SW	171	203	241	363	379	435
	NS	157	191	229	349	365	419
	KL	89	125	163	285	301	353
ActivateTaskset	SW	91	457	501	99	463	519
	NS	79	441	485	85	447	501
	KL	43	399	441	51	409	477
	SW2	91	455	501	97	463	517
	NS2	77	439	485	85	447	501
	KL2	43	397	439	53	409	477
ChainTaskset	SWL	253	625	697	301	663	745
	SWH	311	685	755	359	719	805

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NSL	253	625	695	301	661	745
	NSH	303	677	749	351	713	799
GetTasksetRef		21	23	21	21	23	21
MergeTaskset		61	61	61	61	61	61
AssignTaskset		19	19	19	19	19	19
RemoveTaskset		61	61	61	61	61	61
TestSubTaskset		71	71	71	71	71	71
TestEquivalentTaskset		71	69	71	71	69	71
TickSchedule	SW	157	557	599	211	579	643
	NS	141	535	577	189	555	621
	KL	105	495	537	151	517	581
	SW2	155	557	599	211	567	635
	NS2	139	535	577	189	545	613
	KL2	103	497	539	151	507	575
AdvanceSchedule	SW	131	539	581	193	561	625
	NS	113	517	559	171	537	603
	KL	79	481	523	135	503	567
	SW2	131	539	581	193	549	617
	NS2	113	517	559	171	527	595
	KL2	79	483	525	135	493	561
StartSchedule		93	93	95	93	93	95
StopSchedule		79	79	79	79	79	79
GetScheduleStatus		89	89	89	89	89	89
GetScheduleValue		83	83	83	83	83	83
GetScheduleNext		21	21	23	21	21	23
SetScheduleNext		23	23	21	23	23	21
GetArrivalpointDelay		19	19	19	19	19	19
SetArrivalpointDelay		19	19	19	19	19	19
GetArrivalpointTasksetRef		17	17	15	17	17	15
GetArrivalpointNext		19	19	19	19	19	19
SetArrivalpointNext		19	19	19	19	19	19
TestArrivalpointWritable		27	27	29	27	27	29
GetExecutionTime		7	9	7	7	7	9
GetLargestExecutionTime		17	17	17	17	17	17
ResetLargestExecutionTime		15	13	15	15	15	13
GetStackOffset		31	29	31	31	31	29

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	101	133	171	107	123	179
	NS	89	119	159	95	109	165
	KL	53	87	125	59	77	129
TerminateTask	LExt	0	0	0	0	0	0
	H	411	411	417	411	411	419
ChainTask	SWL	563	609	673	619	635	713
	SWH	611	659	723	669	683	765
	NSL	563	609	673	619	637	713
	NSH	601	649	713	659	673	757
Schedule	SW	81	81	91	81	81	91
GetTaskID		31	31	31	31	31	31
GetTaskState		89	87	89	97	97	97
EnableAllInterrupts		33	33	33	33	33	33
DisableAllInterrupts		49	47	47	47	47	47
ResumeAllInterrupts		43	43	43	43	43	43
SuspendAllInterrupts		63	65	65	65	65	65
ResumeOSInterrupts		43	43	43	43	43	43
SuspendOSInterrupts		65	63	63	63	63	63
GetResource	Task	41	43	43	43	41	43
	Combined	57	59	57	59	57	57
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	77	79	77	79	77	77
	Combined	113	113	113	113	113	113
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	107	107	107
	NS	n/a	n/a	n/a	105	105	105
	KL	n/a	n/a	n/a	73	73	73
ClearEvent		n/a	n/a	n/a	65	65	65
GetEvent		n/a	n/a	n/a	25	25	25
WaitEvent	<default>	n/a	n/a	n/a	641	643	685
	fp	n/a	n/a	n/a	649	649	695
GetAlarmBase		69	69	69	69	71	71
GetAlarm		93	93	93	93	93	93

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		127	127	127	127	127	127
SetAbsAlarm		109	109	109	109	109	109
CancelAlarm		73	73	73	73	75	75
InitCounter		67	67	67	67	67	67
GetCounterValue		73	73	73	73	71	71
osek_tick_alarm	<default>	83	81	81	81	81	81
	KL	43	43	43	43	41	41
osek_incr_counter		17	17	17	17	17	17
GetActiveApplicationMode		13	13	13	13	13	13
StartOS		2927	2927	2927	2927	2927	2927
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	51	51	51	51	51	51
InitCOM		9	9	9	11	9	11
CloseCOM		9	11	11	11	11	9
StartCOM		33	33	33	101	103	101
StopCOM		19	19	19	19	19	19
ReadFlag		n/a	n/a	n/a	21	19	21
ResetFlag		n/a	n/a	n/a	15	15	15
ReceiveMessage		69	69	67	255	255	255
GetMessageResource		n/a	n/a	n/a	91	91	91
ReleaseMessageResource		n/a	n/a	n/a	137	137	137
GetMessageStatus		n/a	n/a	n/a	41	41	41
SendMessage	SW	173	205	245	365	381	437
	NS	161	191	231	351	367	421
	KL	93	127	165	287	303	355
ActivateTaskset	SW	93	459	503	99	465	521
	NS	81	441	487	87	449	503
	KL	45	401	441	55	411	479
	SW2	93	457	503	101	465	519
	NS2	79	443	487	87	449	503
	KL2	45	399	443	53	411	479
ChainTaskset	SWL	559	931	1003	609	969	1051
	SWH	611	985	1057	659	1021	1107
	NSL	555	931	1003	609	969	1053
	NSH	601	975	1047	651	1011	1097
GetTasksetRef		23	25	25	23	23	25

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		63	63	63	63	63	63
AssignTaskset		21	21	21	21	21	21
RemoveTaskset		63	63	63	63	63	63
TestSubTaskset		73	73	73	73	73	73
TestEquivalentTaskset		73	71	71	73	73	71
TickSchedule	SW	159	559	601	213	581	645
	NS	141	537	579	191	559	623
	KL	107	499	541	153	519	583
	SW2	159	559	601	213	571	639
	NS2	143	537	579	191	549	617
	KL2	107	497	539	151	509	577
AdvanceSchedule	SW	133	541	583	195	563	627
	NS	115	519	561	173	541	605
	KL	81	485	527	137	505	571
	SW2	135	541	583	195	553	619
	NS2	115	519	561	173	531	597
	KL2	79	485	527	139	495	563
StartSchedule		97	95	95	95	97	97
StopSchedule		81	81	81	81	81	81
GetScheduleStatus		91	91	91	91	91	91
GetScheduleValue		85	85	85	85	85	85
GetScheduleNext		25	23	23	23	25	25
SetScheduleNext		23	25	25	25	23	23
GetArrivalpointDelay		21	21	21	21	21	21
SetArrivalpointDelay		21	21	21	21	21	21
GetArrivalpointTasksetRef		17	19	19	19	17	17
GetArrivalpointNext		21	21	21	21	21	21
SetArrivalpointNext		21	21	21	21	21	21
TestArrivalpointWritable		31	29	29	29	31	31
GetExecutionTime		157	157	157	157	157	157
GetLargestExecutionTime		27	27	27	29	29	27
ResetLargestExecutionTime		25	25	25	25	25	25
GetStackOffset		33	33	33	31	31	33

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	289	325	361	299	315	367
	NS	309	345	381	319	335	389
	KL	243	277	315	253	267	319
TerminateTask	LExt	479	481	487	479	481	485
	H	503	507	511	503	507	513
ChainTask	SWL	815	865	929	873	891	971
	SWH	863	913	977	921	937	1021
	NSL	845	893	957	903	919	999
	NSH	887	937	1003	945	963	1047
Schedule	SW	131	131	141	131	131	141
GetTaskID		39	37	39	39	39	37
GetTaskState		309	311	311	313	313	313
EnableAllInterrupts		43	41	43	43	41	41
DisableAllInterrupts		57	57	57	57	57	57
ResumeAllInterrupts		61	61	61	61	61	61
SuspendAllInterrupts		73	73	73	73	73	73
ResumeOSInterrupts		61	61	61	61	61	61
SuspendOSInterrupts		73	73	73	73	73	73
GetResource	Task	431	429	245	467	465	281
	Combined	219	219	219	255	255	255
	CLEx	251	249	251	285	287	285
ReleaseResource	Task	231	231	231	267	267	267
	Combined	247	247	247	283	283	283
	CLEx	219	217	219	253	255	253
SetEvent	SW	n/a	n/a	n/a	321	321	321
	NS	n/a	n/a	n/a	341	341	339
	KL	n/a	n/a	n/a	289	289	289
ClearEvent		n/a	n/a	n/a	107	107	105
GetEvent		n/a	n/a	n/a	251	251	251
WaitEvent	<default>	n/a	n/a	n/a	741	743	783
	fp	n/a	n/a	n/a	747	749	791
GetAlarmBase		197	197	209	197	197	209
GetAlarm		211	211	223	211	211	223

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		265	265	277	265	263	277
SetAbsAlarm		237	235	247	235	235	247
CancelAlarm		191	191	203	191	191	203
InitCounter		289	287	303	287	287	303
GetCounterValue		187	187	187	187	187	187
osek_tick_alarm	<default>	99	101	99	101	101	99
	KL	39	39	41	39	41	41
osek_incr_counter		15	15	15	15	15	15
GetActiveApplicationMode		11	11	11	11	11	11
StartOS		3027	3027	3027	3027	3027	3027
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	55	55	55	55	55	55
InitCOM		7	7	7	7	9	7
CloseCOM		7	9	9	9	9	9
StartCOM		47	47	47	113	115	113
StopCOM		33	35	35	35	35	35
ReadFlag		n/a	n/a	n/a	33	35	33
ResetFlag		n/a	n/a	n/a	29	29	29
ReceiveMessage		157	159	157	343	343	341
GetMessageResource		n/a	n/a	n/a	389	387	387
ReleaseMessageResource		n/a	n/a	n/a	387	387	387
GetMessageStatus		n/a	n/a	n/a	109	109	109
SendMessage	SW	453	487	525	643	661	711
	NS	473	509	547	663	681	733
	KL	375	407	445	565	583	633
ActivateTaskset	SW	581	857	889	599	853	913
	NS	509	873	915	619	867	927
	KL	437	799	843	457	801	867
	SW2	579	857	889	601	853	911
	NS2	507	875	917	617	865	929
	KL2	437	801	845	455	803	869
ChainTaskset	SWL	1123	1445	1513	1185	1449	1537
	SWH	1171	1483	1555	1233	1487	1609
	NSL	1149	1467	1543	1217	1475	1599
	NSH	1189	1513	1587	1251	1519	1641
GetTasksetRef		227	227	229	227	229	227

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		139	139	139	139	139	139
AssignTaskset		83	83	81	83	81	83
RemoveTaskset		139	139	139	139	139	139
TestSubTaskset		153	153	155	153	155	153
TestEquivalentTaskset		153	153	153	153	153	153
TickSchedule	SW	217	993	1039	649	1015	1077
	NS	231	1007	1049	661	1027	1091
	KL	155	939	981	595	959	1023
	SW2	217	993	1039	649	997	1061
	NS2	231	1007	1049	663	1009	1075
	KL2	155	939	981	595	939	1007
AdvanceSchedule	SW	197	983	1027	639	1003	1067
	NS	227	995	1041	653	1017	1079
	KL	143	931	977	587	953	1015
	SW2	195	983	1027	639	985	1051
	NS2	227	995	1041	653	999	1063
	KL2	145	931	977	589	933	999
StartSchedule		155	153	153	153	153	155
StopSchedule		121	123	123	123	123	121
GetScheduleStatus		135	135	135	135	135	135
GetScheduleValue		127	125	125	125	125	127
GetScheduleNext		55	55	55	55	55	55
SetScheduleNext		83	83	83	83	83	83
GetArrivalpointDelay		63	61	61	61	61	63
SetArrivalpointDelay		73	73	73	73	73	73
GetArrivalpointTasksetRef		55	53	53	53	53	55
GetArrivalpointNext		57	57	57	57	57	57
SetArrivalpointNext		85	85	85	85	85	85
TestArrivalpointWritable		59	61	61	61	61	59
GetExecutionTime		183	181	181	181	181	183
GetLargestExecutionTime		217	215	215	215	215	217
ResetLargestExecutionTime		209	209	209	209	209	209
GetStackOffset		29	31	31	31	31	29

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	43	43	43	43	43	43
	Cat 2	87	109	109	109	109	109

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	43	43	43	43	43	43
	Cat 2	287	303	303	303	303	303

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	43	43	43	43	43	43
	Cat 2	287	303	303	303	303	303

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.

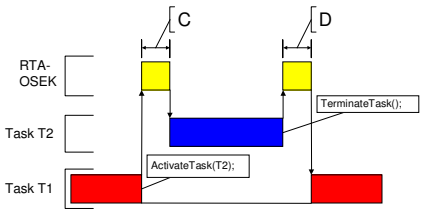

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

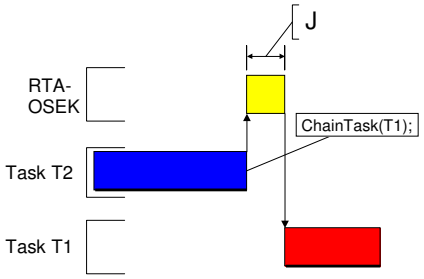

Figure 2: Task Chaining

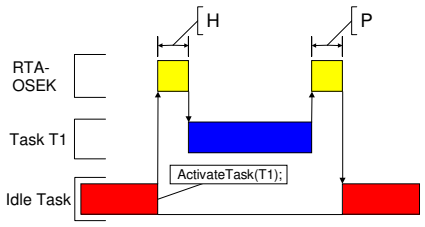

Figure 3: Task Activation from Idle Task

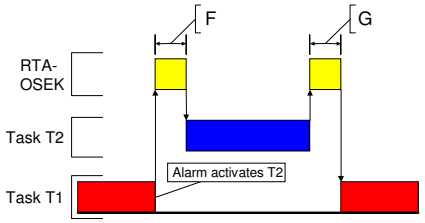

Figure 4: Task Activation from an Alarm

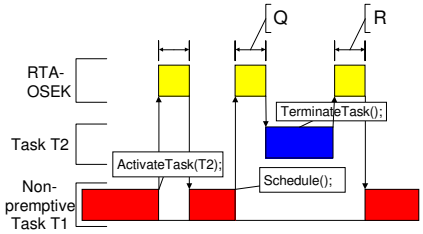

Figure 5: Non-Premptive Task Calls Schedule()

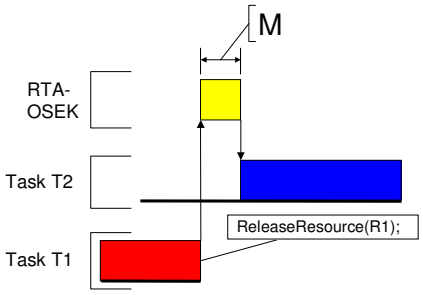

Figure 6: Blocked Task Activated by ReleaseResource()

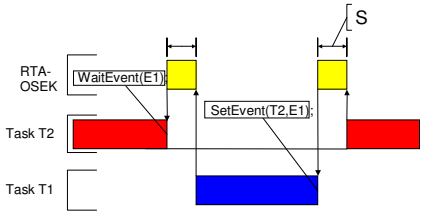

Figure 7: Waiting Task Activated by SetEvent()

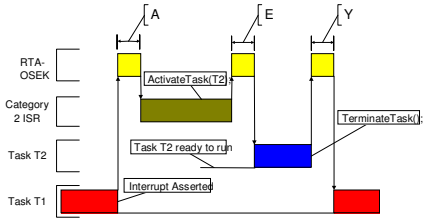

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	81	127	149	81	127	153
Figure 1: D	Heavy, Basic/Extended	133	173	201	177	179	205
ChainTask	Light, Basic	199	257	321	205	255	333
Figure 2: J	Heavy, Basic/Extended	469	563	657	519	571	677
Pre-emption	Light, Basic	181	235	313	189	235	327
Figure 1: C	Heavy, Basic/Extended	239	281	359	295	307	399
From idle task	Light, Basic	181	235	313	189	235	327
Figure 3: H	Heavy, Basic/Extended	239	283	359	295	309	401
Triggered by alarm	Light, Basic	267	321	397	275	323	415
Figure 4: F	Heavy, Basic/Extended	325	367	443	379	393	487
Schedule	Light, Basic	161	185	233	161	183	233
Figure 5: Q	Heavy, Basic/Extended	219	231	279	267	267	315
Release resource	Light, Basic	173	197	235	173	197	233
Figure 6: M	Heavy, Basic/Extended	231	243	281	279	279	317
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	421	421	531
From category 2 ISR	Light, Basic	131	175	213	153	175	215
Figure 8: E	Heavy, Basic/Extended	189	223	261	259	259	297

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	363	397	421	363	397	423
Figure 1: D	Heavy, Basic/Extended	411	435	465	443	445	469
ChainTask	Light, Basic	517	557	623	523	557	635
Figure 2: J	Heavy, Basic/Extended	1059	1123	1217	1095	1129	1239
Pre-emption	Light, Basic	399	437	519	407	437	533
Figure 1: C	Heavy, Basic/Extended	445	489	569	503	515	611
From idle task	Light, Basic	399	437	519	407	437	533
Figure 3: H	Heavy, Basic/Extended	445	489	569	503	517	611
Triggered by alarm	Light, Basic	485	523	603	493	525	621
Figure 4: F	Heavy, Basic/Extended	531	575	653	589	603	699
Schedule	Light, Basic	379	387	439	379	385	439
Figure 5: Q	Heavy, Basic/Extended	425	439	489	475	475	527
Release resource	Light, Basic	399	407	449	399	407	447
Figure 6: M	Heavy, Basic/Extended	445	459	499	495	495	535
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	607	607	709
From category 2 ISR	Light, Basic	645	669	711	661	669	711
Figure 8: E	Heavy, Basic/Extended	691	721	761	757	757	799

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	475	509	533	475	509	533
Figure 1: D	Heavy, Basic/Extended	503	529	559	535	539	563
ChainTask	Light, Basic	769	813	877	779	813	891
Figure 2: J	Heavy, Basic/Extended	1405	1473	1569	1445	1483	1587
Pre-emption	Light, Basic	581	619	699	589	619	713
Figure 1: C	Heavy, Basic/Extended	627	671	749	685	699	791
From idle task	Light, Basic	581	619	699	589	619	713
Figure 3: H	Heavy, Basic/Extended	627	671	749	685	699	793
Triggered by alarm	Light, Basic	685	723	805	695	725	821
Figure 4: F	Heavy, Basic/Extended	731	775	853	791	805	899
Schedule	Light, Basic	419	427	479	419	425	477
Figure 5: Q	Heavy, Basic/Extended	465	479	529	515	515	567
Release resource	Light, Basic	531	539	581	567	575	617
Figure 6: M	Heavy, Basic/Extended	577	591	631	663	665	705
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	841	841	941
From category 2 ISR	Light, Basic	671	693	735	687	693	735
Figure 8: E	Heavy, Basic/Extended	717	745	787	783	783	823

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	72	72	76	72	72	76
BCC1 lightweight, floating-point	80	80	84	80	80	84
BCC1 heavyweight, integer	124	124	128	124	124	128
BCC1 heavyweight, floating-point	124	124	128	124	124	128
BCC2 lightweight, integer	n/a	84	88	n/a	84	88
BCC2 lightweight, floating-point	n/a	84	88	n/a	84	88
BCC2 heavyweight, integer	n/a	132	136	n/a	132	136
BCC2 heavyweight, floating-point	n/a	132	136	n/a	132	136
ECC1 heavyweight, integer	n/a	n/a	n/a	156	156	160
ECC1 heavyweight, floating-point	n/a	n/a	n/a	156	156	160
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	160
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	160
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	80	80	80	80	80	80
BCC1 lightweight, floating-point	88	88	88	88	88	88
BCC1 heavyweight, integer	132	132	132	132	132	132
BCC1 heavyweight, floating-point	132	132	132	132	132	132
BCC2 lightweight, integer	n/a	92	92	n/a	92	92
BCC2 lightweight, floating-point	n/a	92	92	n/a	92	92
BCC2 heavyweight, integer	n/a	140	140	n/a	140	140
BCC2 heavyweight, floating-point	n/a	140	140	n/a	140	140
ECC1 heavyweight, integer	n/a	n/a	n/a	164	164	164
ECC1 heavyweight, floating-point	n/a	n/a	n/a	164	164	164
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	164
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	164

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations	Application Uses					
	No			Yes		
	No		Yes	No		Yes
	No	Yes		No	Yes	
Pre- and Post-Task hooks not used						
Task type						
BCC1 lightweight, integer	96	96	100	96	96	100
BCC1 lightweight, floating-point	100	100	104	100	100	104
BCC1 heavyweight, integer	148	148	152	148	148	152
BCC1 heavyweight, floating-point	148	148	152	148	148	152
BCC2 lightweight, integer	n/a	104	112	n/a	104	112
BCC2 lightweight, floating-point	n/a	104	112	n/a	104	112
BCC2 heavyweight, integer	n/a	160	164	n/a	160	164
BCC2 heavyweight, floating-point	n/a	160	164	n/a	160	164
ECC1 heavyweight, integer	n/a	n/a	n/a	184	184	188
ECC1 heavyweight, floating-point	n/a	n/a	n/a	184	184	188
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	188
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	188
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	100	100	104	100	100	104
BCC1 lightweight, floating-point	104	104	108	104	104	108
BCC1 heavyweight, integer	152	152	156	152	152	156
BCC1 heavyweight, floating-point	152	152	156	152	152	156
BCC2 lightweight, integer	n/a	108	116	n/a	108	116
BCC2 lightweight, floating-point	n/a	108	116	n/a	108	116
BCC2 heavyweight, integer	n/a	164	168	n/a	164	168
BCC2 heavyweight, floating-point	n/a	164	168	n/a	164	168
ECC1 heavyweight, integer	n/a	n/a	n/a	188	188	192
ECC1 heavyweight, floating-point	n/a	n/a	n/a	188	188	192
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	192
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	192

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	96	96	100	96	96	100
BCC1 lightweight, floating-point	100	100	104	100	100	104
BCC1 heavyweight, integer	148	148	152	148	148	152
BCC1 heavyweight, floating-point	148	148	152	148	148	152
BCC2 lightweight, integer	n/a	104	112	n/a	104	112
BCC2 lightweight, floating-point	n/a	104	112	n/a	104	112
BCC2 heavyweight, integer	n/a	160	164	n/a	160	164
BCC2 heavyweight, floating-point	n/a	160	164	n/a	160	164
ECC1 heavyweight, integer	n/a	n/a	n/a	192	192	196
ECC1 heavyweight, floating-point	n/a	n/a	n/a	192	192	196
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	196
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	196
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	100	100	104	100	100	104
BCC1 lightweight, floating-point	104	104	108	104	104	108
BCC1 heavyweight, integer	152	152	156	152	152	156
BCC1 heavyweight, floating-point	152	152	156	152	152	156
BCC2 lightweight, integer	n/a	108	116	n/a	108	116
BCC2 lightweight, floating-point	n/a	108	116	n/a	108	116
BCC2 heavyweight, integer	n/a	164	168	n/a	164	168
BCC2 heavyweight, floating-point	n/a	164	168	n/a	164	168
ECC1 heavyweight, integer	n/a	n/a	n/a	196	196	200
ECC1 heavyweight, floating-point	n/a	n/a	n/a	196	196	200
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	200
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	200

5 Compatibility with Pre-v5.00 Kernels

5.1 Updating the Application Version

To convert an existing OIL configuration file that uses the v3.1 kernel library to use the v5.00 kernel libraries, 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 kernel libraries. This process can be reversed to move back to earlier kernel versions.

5.2 Changes between versions 3.1 and 5.00

The following modifications have been made between versions 3.1 and 5.00:

- The compiler version used to build and test the libraries has been modified from v2.30 to v4.4.1
- The performance of the kernel has been optimized; especially as the kernel only uses the SVC stack the kernel now only monitors the SVC stack use
- Support has been added to attach an interrupt handler to the "phantom" interrupt source
- RTA-TRACE is now fully now supported when using an RS232 link
- ORTI debugging is now supported for either the Lauterbach Trace32 debugger or a generic ORTI debugger
- The default CIM interrupt de-multiplexer can now be replaced with a user provided alternative

The issue that occurred when generating a vector table where the RTA-OSEK interrupt vector de-multiplexers `os_fiq_entry()` and `os_irq_entry()` were not located on the FIQ and IRQ CPU interrupt lines when only one CIM interrupt was in the application has been corrected

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.