RTA-OSEK

Binding Manual: Virtex405/Diab

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.
3F, Samseung Bldg. 61-1
Yangjae-dong, Seocho-gu
Seoul

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: RM00066-003

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 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 port specific information for the Virtex405/Diab implementation of LiveDevices' RTA-OSEK.

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?

It is assumed that you are a developer. 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.

In this guide you'll see that program code, header file names, C type names, C functions 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

In this chapter, you'll see the important details that you need to know about RTA-OSEK and your toolchain. A port of the RTA-OSEK Component is specific to both the target hardware *and* the compiler toolchain. You must make sure that you build your application with this toolchain.

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

The Virtex405/Diab supports the single flat memory model supported by the Wind River Systems, Inc. (Diab) toolchain. This toolchain supports the Embedded Application Binary Interface, EABI.

2.1 Compiler

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

Vendor	Wind River Systems, Inc.
Compiler	Wind Power (Diab) C/C++ Compiler for PowerPC
Version	5.2.1.0

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

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

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

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

Option	Description
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.
`-g0`	Turn debug mode off.

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

Option	Description
`-g`	Debugging must be disabled.

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. PPC405EN:simple).

2.2 Assembler

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

Vendor	Wind River Systems, Inc.
Assembler	Wind Power (Diab) Assembler for the PowerPC
Version	5.2.1.0

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

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

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

2.3 Linker/Locator

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

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

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

Sections	ROM/RAM	Description
`os_pid`	ROM	RTA-OSEK read-only data
`os_pird`	ROM	RTA-OSEK initialization data
`os_pnird`	ROM	RTA-OSEK near initialization data.
`os_vectbl`	ROM	Vector table (if generated by RTA-OSEK). Should be aligned on a 64KByte boundary
`os_text`	ROM	RTA-OSEK code section.
`os_pir`	RAM	RTA-OSEK initialized data. Must be initialized during C-startup. Can be located in 'far' RAM.
`os_pnir`	RAM	RTA-OSEK near initialized data. Must be initialized during C-startup. Must be located in the compiler SDA (i.e. near addressing is used).
`os_pur`	RAM	RTA-OSEK uninitialized data. Must be zeroed during C-startup.

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

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

Important: The symbol `__os_intc_base` must be defined in the linker command file equal to the base address of the OPB interrupt controller; this is demonstrated in the supplied example application.

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 (`.elf`) 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. 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 *Xilinx PowerPC Processor Reference Guide*.

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

IPL	INTC_EIR	TCR[TIE]	TCR[PIE]	Description
0	0xFFFFFFFF	1	1	User level
1	0xFFFFFFFF	1	0	PIT Category 2 interrupt
2	0xFFFFFFFF	0	0	FIT Category 1 or 2 interrupt
3	0x7FFFFFFF	0	0	INT31 Category 1 or 2 interrupt
-	-	-	-	-
33	0x00000001	0	0	INT1 Category 1 or 2 interrupt
34	0x00000000	0	0	INT0 Category 1 or 2 interrupt only
35	0x00000000	0	0	Non-Maskable CPU interrupts (i.e. MSR[EE] bit is clear)

3.1.2 Interrupt Vectors

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

Vector	Legality
0x0 to 0x1F	The INTC vectors can handle Category 1 and 2 ISRs
0x500, 0x1000 and 0x1010	The CPU vectors that can handle Category 1 and 2 ISRs
0x100, 0x200, 0x300, 0x400, 0x600, 0x700, 0x800, 0xC00, 0xF20, 0x1020, 0x1100, 0x1200, 0x2000	The CPU vectors that can handle Category 1 ISRs only

The valid base addresses for the vector table are:

Base Address	Notes
EVPR	The base address of the vector table should be aligned to a 64 Kbyte boundary.

3.1.3 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Wind River Systems, Inc. C compiler can generate appropriate interrupt handling code for a C function decorated with the `__interrupt__` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

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

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

The following table shows the syntax for labels attached to RTA-OSEK Category 2 interrupt handlers (VVVV represents the 4 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
`0xVVVV`	`os_wrapper_VVVV`
eg : 0x1010	`os_wrapper_1010`

3.1.6 Processor Mode

RTA-OSEK operates in the processor's supervisor mode and expects applications to do so also.

Important: The application should not set the "Problem State" bit of the Machine State Register, MSR[PR].

3.1.7 INTC and CPU vector offset mapping

The PPC405 has two exception sources the CPU and the INTC Interrupt Controller.

CPU interrupts and exceptions: The addresses of the handler functions for the CPU exceptions are formed by combining the contents of the EVPR register and the offset specific to the exception source. The CPU vector table is generated within the file `osgen.s`. The start of the CPU vector table has the label `os_cpu_vect`, which should be used to initialize the EVPR register; this is demonstrated in the example application.

INTC interrupts and exceptions: The 32 INTC channels are characterized in an integer array `os_intc_isrs` generated by RTA-OSEK, which contains the addresses of the interrupt handlers for ISRs on these vectors. When an INTC interrupt is triggered RTA-OSEK uses the IVR value to access `os_intc_isrs` in order to branch to the appropriate interrupt handler.

The offset addresses, shown in Section 3.1.2, illustrate the direct mapping between the range of entries and the interrupt sources that should be used in RTA-OSEK.

Important: RTA-OSEK for the Virtex405/Diab is designed to be used with the Xilinx OPB interrupt controller not the Xilinx DCR interrupt controller.

3.1.8 Default Interrupt Support

The default interrupt is only supported for CPU interrupts and exceptions and should be decorated by the `__interrupt__` function qualifier.

3.1.9 Handling multiple INTC interrupt sources

The INTC allows multiple interrupt sources to be processed and handled by the PPC405 External Exception (EE) or Critical Exception (CE) line. In the EE INTC case RTA-OSEK supports these multiple INTC interrupts using the interrupt handler function `os_ee_entry()`. This is a multiple interrupt handler entry point and references the INTC IVR register to determine the interrupt source.

If RTA-OSEK generates a vector table and multiple INTC interrupts are contained within the application OIL configuration file, this function is automatically located on CPU vector table address 0x500. Alternatively if the

user provides the vector table for an application with multiple INTC interrupts, this function should be bound to address 0x500.

When multiple INTC 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_ee_entry()`.

3.1.10 Handling single INTC interrupt sources

When an application only contains a single INTC interrupt source the INTC hardware does not need to be referenced. Instead, the interrupt can be bound directly to the CPU EE (0x500) or CE (0x100) vector, consequently reducing the interrupt entry time. Single Category 2 IRQ interrupts should be bound to vector 0x500 in RTA-OSEK. The entry function that occurs in the vector table is `os_wrapper_0500()`. If a user generated vector table is used with a single Category 2 IRQ interrupt this should be bound to the EE entry.

3.1.11 Location of the CPU vector table

The CPU vector table uses the PowerPC relative branch `b` assembly instruction to branch to an interrupt handler. The range of the supported branch is 32 Mbytes. If the CPU vector table is located outside of the valid branch range to a handler function the compiler inserts a "branch island". The use of a branch island during the initial interrupt branch will corrupt the general purpose registers contents and should be avoided.

3.2 Register Settings

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

Register	Notes
EVPR	Interrupt vector table base address.
MSR[EE]	The EE bit should be set to enable External Interrupts.
MSR[PR]	The PR bit should not be set as RTA-OSEK expects that the processor always operates at supervisor level.
INTC_MER[ME]	The ME bit should be set to enable IRQ interrupts in the INTC.

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

Register	Notes
MSR[EE]	The external interrupt enable bit should not be manipulated by the user after calling StartOS().
INTC_IER	The interrupt enable bits of the INTC should not be manipulated by the user after calling StartOS().
TCR[PIE & TIE]	The PIT and FIT interrupt enable bits should not be manipulated by the user after calling StartOS().
INTC_MER[ME]	The master IRQ enable bit of the INTC should not be manipulated by the user after calling StartOS().

3.3 Stack Usage

3.3.1 Number of Stacks

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

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

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): 80

Timing

API max usage (bytes): 80

Extended

API max usage (bytes): 96

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

3.3.3 Stack discipline

RTA-OSEK adheres to the EABI requirements for stack discipline, in particular that the stack pointer (R1) is adjusted only once in each routine, a back-link is maintained, and the stack pointer is kept aligned to a 16-byte boundary. During start-up the user's application code should set R1 to a suitable value, in on-chip or external RAM.

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

The linker control file will typically need to include the line (demonstrated in the example application)

```
OS_SP_INIT = __SP_INIT;
```

4 Parameters of Implementation

This chapter provides detailed information on the functionality, performance and memory demands of the RTA-OSEK Component.

The RTA-OSEK Component is highly scalable. As a result, different figures will be obtained when your application uses different sets of features. These feature-sets give six classes of RTA-OSEK, depending on whether your application uses events, shared task priorities and/or multiple (queued) task activations. You should identify which class your application belongs to and then use the figures from the appropriate column in the table.

The following hardware was used to take the measurements in this chapter:

Processor	**PPC405**
Clock speed (MHz)	300
Code memory	Block RAM (instruction cache enabled)
Read-only data memory	Block RAM
Read-write data memory	Block RAM

4.1 Functionality

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

Configuration	**Application Uses**					
Events	**No**			**Yes**		
Shared Task Priorities	**No**		**Yes**	**No**		**Yes**
Multiple Task Activations	**No**	**Yes**		**No**	**Yes**	
Maximum number of tasks	32	32	32	32	32	32
Maximum number of not suspended tasks	32	32	32	32	32	32
Maximum number of priorities	32	32	32	32	32	32
Number of tasks per priority (for BCC2 and ECC2)	n/a	32	32	n/a	32	32
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	32	32	32
Limits for the number of alarm objects (per system / per task)	not limited by RTA-OSEK					
Limits for the number of standard resources (per system)	255	255	255	255	255	255
Limits for the number of internal resources (per system)	not limited by RTA-OSEK					
Limits for the number of nested resources (per system / per task)	255	255	255	255	255	255
Limits for the number of application modes	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	28	28	28	28	28	28
	ROM	154	154	154	154	154	154
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	48	48	48	48	48	48
	ROM	226	226	226	226	226	226
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	74	74	74	74	74	74
	ROM	272	272	272	272	272	272
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	40	40	40	40	40	40
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	44	44	44	44	44	44
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	48	56	n/a	48	56
ECC1, Integer task	RAM	n/a	n/a	n/a	268	268	268
	ROM	n/a	n/a	n/a	64	64	64
ECC1, floating-point task	RAM	n/a	n/a	n/a	270	270	270
	ROM	n/a	n/a	n/a	64	64	64
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	270
	ROM	n/a	n/a	n/a	n/a	n/a	72
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	272
	ROM	n/a	n/a	n/a	n/a	n/a	72
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	52	52	52	52	52	52
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	88	88	88	88	88	88
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
Counter	RAM	4	4	4	4	4	4
	ROM	68	68	68	68	68	68
Message	RAM	11	11	11	51	51	51
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
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
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	52	52	52	52	52	52
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	56	56	56	56	56	56
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	60	68	n/a	60	68
ECC1, Integer task	RAM	n/a	n/a	n/a	280	280	280
	ROM	n/a	n/a	n/a	76	76	76
ECC1, floating-point task	RAM	n/a	n/a	n/a	282	282	282
	ROM	n/a	n/a	n/a	76	76	76
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	282
	ROM	n/a	n/a	n/a	n/a	n/a	84
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	284
	ROM	n/a	n/a	n/a	n/a	n/a	84

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	116	116	116	116	116	116
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	148	148	148	148	148	148
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
Counter	RAM	4	4	4	4	4	4
	ROM	68	68	68	68	68	68
Message	RAM	11	11	11	51	51	51
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
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

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	64	64	64	64	64	64
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	64	64	64	64	64	64
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	284	284	284
	ROM	n/a	n/a	n/a	84	84	84
ECC1, floating-point task	RAM	n/a	n/a	n/a	286	286	286
	ROM	n/a	n/a	n/a	84	84	84
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	286
	ROM	n/a	n/a	n/a	n/a	n/a	92
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	288
	ROM	n/a	n/a	n/a	n/a	n/a	92
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	128	128	128	128	128	128
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	160	160	160	160	160	160
Resource	RAM	16	16	16	16	16	16
	ROM	28	28	28	28	28	28
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	16	16	16	16	16	16
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	52	52	52	52	52	52
Counter	RAM	4	4	4	4	4	4
	ROM	72	72	72	72	72	72
Message	RAM	11	11	11	51	51	51
	ROM	24	24	24	60	60	60
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	16	16	16	16	16	16
	ROM	28	28	28	28	28	28

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
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.

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

Standard

Configuration			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	304	396	448	312	404	480
	NS		284	352	404	292	360	436
	KL	2	88	176	232	96	184	260

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	20	20	20	20	20	20
ChainTask	SWL	1, 8	252	356	404	260	364	444
	SWH	1, 9	316	416	456	324	424	496
	NSL	8	252	356	404	260	364	444
	NSH	9	324	416	464	332	424	472
Schedule			248	248	296	248	248	296
GetTaskID			44	44	44	44	44	44
GetTaskState			240	240	240	264	264	264
EnableAllInterrupts			76	76	76	76	76	76
DisableAllInterrupts			116	116	116	116	116	116
ResumeAllInterrupts			100	100	100	100	100	100
SuspendAllInterrupts			148	148	148	148	148	148
ResumeOSInterrupts			92	92	92	92	92	92
SuspendOSInterrupts			192	192	192	192	192	192
GetResource	Task	7	32	32	40	32	32	40
	Combined	6	236	236	236	236	236	236
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	224	224	224	224	224	224
	Combined	6	448	448	448	448	448	448
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	288	288	396
	NS		n/a	n/a	n/a	252	252	336
	NS1i	10	n/a	n/a	n/a	200	n/a	n/a
	KL	2	n/a	n/a	n/a	76	76	188
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	148	148	148
GetEvent			n/a	n/a	n/a	20	20	20
WaitEvent	<default>		n/a	n/a	n/a	396	396	720
	fp	11	n/a	n/a	n/a	440	440	812
	1i	10	n/a	n/a	n/a	24	n/a	n/a
GetAlarmBase			212	212	212	212	212	212
GetAlarm			244	244	244	244	244	244
SetRelAlarm			288	288	288	288	288	288
SetAbsAlarm			300	300	300	300	300	300
CancelAlarm			232	232	232	232	232	232
InitCounter			204	204	204	204	204	204

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			232	232	232	232	232	232
osek_tick_alarm	<default>		240	240	240	240	240	240
	KL	2	68	68	68	68	68	68
osek_incr_counter			60	60	60	60	60	60
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			380	380	380	380	380	380
ShutdownOS	NoHook	12	112	112	112	112	112	112
	Hook	13	136	136	136	136	136	136
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			60	60	60	60	60	60
StopCOM			32	32	32	32	32	32
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	200	200	200	372	372	372
	CCCB	15	372	372	372	372	372	372
GetMessageResource			88	88	88	88	88	88
ReleaseMessageResource			80	80	80	80	80	80
GetMessageStatus			76	76	76	76	76	76
SendMessage	SW CCCA	1, 14	236	236	236	448	448	448
	SW CCCB	1, 15	424	424	424	448	448	448
	NS CCCA	14	236	236	236	448	448	448
	NS CCCB	15	424	424	424	448	448	448
	KL CCCA	2, 14	116	116	116	324	324	324
	KL CCCB	2, 15	300	300	300	324	324	324
main_dispatch	NoHook	12	260	260	320	260	260	320
	Hook	13	308	308	368	308	308	368
sub_dispatch	B1LF	19	48	48	48	48	48	48
	B1HI	20	172	172	172	172	172	172
	B1HF	21	180	180	180	180	180	180
	B2LI	22	n/a	152	192	n/a	152	192
	B2LF	23	n/a	156	196	n/a	156	196
	B2HI	24	n/a	336	424	n/a	336	424
	B2HF	25	n/a	344	432	n/a	344	432
	E1HI	26	n/a	n/a	n/a	648	648	736
	E1HF	27	n/a	n/a	n/a	656	656	744
	E2HI	28	n/a	n/a	n/a	n/a	n/a	736

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	E2HF	29	n/a	n/a	n/a	n/a	n/a	744
ErrorHook support		16	80	80	80	80	80	80
	ServiceID	17	88	88	88	88	88	88
	Parameters	18	108	108	108	108	108	108
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	n/a	n/a	n/a	n/a	n/a	n/a
Timing_termination		4	n/a	n/a	n/a	n/a	n/a	n/a
ActivateTaskset	SW	1	232	312	372	240	332	408
	NS		188	268	312	196	296	348
	KL	2	24	100	180	32	120	212
ChainTaskset	SWL	1, 8	104	180	264	104	192	292
	SWH	1, 9	188	268	328	188	280	356
	NSL	8	104	180	264	104	192	292
	NSH	9	180	260	320	180	272	348
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			196	196	196	196	196	196
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			196	196	196	196	196	196
TestSubTaskset			208	208	208	208	208	208
TestEquivalentTaskset			204	204	204	204	204	204
TickSchedule	SW	1	320	328	328	328	328	328
	NS		276	280	280	280	280	280
	KL	2	112	152	152	152	152	152
AdvanceSchedule	SW	1	308	304	304	304	304	304
	NS		256	256	256	256	256	256
	KL	2	128	136	136	136	136	136
StartSchedule			236	236	236	236	236	236
StopSchedule			208	208	208	208	208	208
GetScheduleStatus			260	260	260	260	260	260
GetScheduleValue			220	220	220	220	220	220
GetScheduleNext			20	20	20	20	20	20
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			12	12	12	12	12	12
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			12	12	12	12	12	12

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestArrivalpointWritable			56	56	56	56	56	56
GetExecutionTime			8	8	8	8	8	8
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			8	8	8	8	8	8
GetStackOffset			20	20	20	20	20	20
Stack manipulation			84	84	84	84	84	84
Interrupt support			296	296	296	296	296	296
Interrupt support			344	344	344	344	344	344

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	304	396	448	312	404	480
	NS		284	352	404	292	360	436
	KL	2	88	176	232	96	184	260
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	20	20	20	20	20	20
ChainTask	SWL	1, 8	252	356	404	260	364	444
	SWH	1, 9	316	416	456	324	424	496
	NSL	8	252	356	404	260	364	444
	NSH	9	324	416	464	332	424	472
Schedule			276	276	324	276	276	324
GetTaskID			44	44	44	44	44	44
GetTaskState			240	240	240	264	264	264
EnableAllInterrupts			76	76	76	76	76	76
DisableAllInterrupts			116	116	116	116	116	116
ResumeAllInterrupts			100	100	100	100	100	100
SuspendAllInterrupts			148	148	148	148	148	148
ResumeOSInterrupts			92	92	92	92	92	92
SuspendOSInterrupts			192	192	192	192	192	192
GetResource	Task	7	32	32	40	32	32	40
	Combined	6	236	236	236	236	236	236

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	252	252	252	252	252	252
	Combined	6	504	504	504	504	504	504
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	288	288	396
	NS		n/a	n/a	n/a	252	252	336
	NS1i	10	n/a	n/a	n/a	200	n/a	n/a
	KL	2	n/a	n/a	n/a	76	76	188
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	148	148	148
GetEvent			n/a	n/a	n/a	20	20	20
WaitEvent	<default>		n/a	n/a	n/a	552	552	868
	fp	11	n/a	n/a	n/a	596	596	968
	1i	10	n/a	n/a	n/a	220	n/a	n/a
GetAlarmBase			212	212	212	212	212	212
GetAlarm			244	244	244	244	244	244
SetRelAlarm			288	288	288	288	288	288
SetAbsAlarm			300	300	300	300	300	300
CancelAlarm			232	232	232	232	232	232
InitCounter			204	204	204	204	204	204
GetCounterValue			232	232	232	232	232	232
osek_tick_alarm	<default>		240	240	240	240	240	240
	KL	2	68	68	68	68	68	68
osek_incr_counter			60	60	60	60	60	60
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			432	432	432	432	432	432
ShutdownOS	NoHook	12	112	112	112	112	112	112
	Hook	13	136	136	136	136	136	136
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			60	60	60	60	60	60
StopCOM			32	32	32	32	32	32
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	200	200	200	372	372	372
	CCCB	15	372	372	372	372	372	372
GetMessageResource			88	88	88	88	88	88

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
ReleaseMessageResource			80	80	80	80	80	80
GetMessageStatus			76	76	76	76	76	76
SendMessage	SW CCCA	1, 14	236	236	236	448	448	448
	SW CCCB	1, 15	424	424	424	448	448	448
	NS CCCA	14	236	236	236	448	448	448
	NS CCCB	15	424	424	424	448	448	448
	KL CCCA	2, 14	116	116	116	324	324	324
	KL CCCB	2, 15	300	300	300	324	324	324
main_dispatch	NoHook	12	248	248	308	248	248	308
	Hook	13	300	300	360	300	300	360
sub_dispatch	B1LF	19	36	36	36	36	36	36
	B1HI	20	132	132	132	132	132	132
	B1HF	21	140	140	140	140	140	140
	B2LI	22	n/a	80	124	n/a	80	124
	B2LF	23	n/a	84	128	n/a	84	128
	B2HI	24	n/a	180	272	n/a	180	272
	B2HF	25	n/a	188	280	n/a	188	280
	E1HI	26	n/a	n/a	n/a	632	632	720
	E1HF	27	n/a	n/a	n/a	640	640	728
	E2HI	28	n/a	n/a	n/a	n/a	n/a	720
	E2HF	29	n/a	n/a	n/a	n/a	n/a	728
ErrorHook support		16	80	80	80	80	80	80
	ServiceID	17	88	88	88	88	88	88
	Parameters	18	108	108	108	108	108	108
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	160	160	160	160	160	160
Timing_termination		4	172	172	172	172	172	172
ActivateTaskset	SW	1	232	312	372	240	332	408
	NS		188	268	312	196	296	348
	KL	2	24	100	180	32	120	212
ChainTaskset	SWL	1, 8	104	180	264	104	192	292
	SWH	1, 9	188	268	328	188	280	356
	NSL	8	104	180	264	104	192	292
	NSH	9	180	260	320	180	272	348
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			196	196	196	196	196	196
AssignTaskset			16	16	16	16	16	16

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
RemoveTaskset			196	196	196	196	196	196
TestSubTaskset			208	208	208	208	208	208
TestEquivalentTaskset			204	204	204	204	204	204
TickSchedule	SW	1	320	328	328	328	328	328
	NS		276	280	280	280	280	280
	KL	2	112	152	152	152	152	152
AdvanceSchedule	SW	1	308	304	304	304	304	304
	NS		256	256	256	256	256	256
	KL	2	128	136	136	136	136	136
StartSchedule			236	236	236	236	236	236
StopSchedule			208	208	208	208	208	208
GetScheduleStatus			260	260	260	260	260	260
GetScheduleValue			220	220	220	220	220	220
GetScheduleNext			20	20	20	20	20	20
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			12	12	12	12	12	12
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			12	12	12	12	12	12
TestArrivalpointWritable			56	56	56	56	56	56
GetExecutionTime			268	268	268	268	268	268
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			20	20	20	20	20	20
GetStackOffset			20	20	20	20	20	20
Stack manipulation			84	84	84	84	84	84
Interrupt support			296	296	296	296	296	296
Interrupt support			344	344	344	344	344	344

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	500	596	644	508	604	680
	NS		596	692	740	604	700	776
	KL	2	312	408	456	320	416	492
TerminateTask	LExt	3	248	248	248	248	248	248
	H	5	300	300	300	300	300	300
ChainTask	SWL	1, 8	552	660	704	560	668	748
	SWH	1, 9	604	704	744	612	712	780
	NSL	8	668	776	820	676	784	864
	NSH	9	712	808	852	720	816	892
Schedule			484	484	532	484	484	532
GetTaskID			68	68	68	68	68	68
GetTaskState			452	452	452	460	460	460
EnableAllInterrupts			100	100	100	100	100	100
DisableAllInterrupts			140	140	140	140	140	140
ResumeAllInterrupts			192	192	192	192	192	192
SuspendAllInterrupts			172	172	172	172	172	172
ResumeOSInterrupts			184	184	184	184	184	184
SuspendOSInterrupts			208	208	208	208	208	208
GetResource	Task	7	720	720	660	720	720	660
	Combined	6	756	756	756	756	756	756
	CLEx	3	628	628	628	628	628	628
ReleaseResource	Task	7	664	664	664	664	664	664
	Combined	6	904	904	904	904	904	904
	CLEx	3	632	632	632	632	632	632
SetEvent	SW	1	n/a	n/a	n/a	560	560	684
	NS		n/a	n/a	n/a	652	652	776
	NS1i	10	n/a	n/a	n/a	468	n/a	n/a
	KL	2	n/a	n/a	n/a	392	392	516
	KL1i	2, 10	n/a	n/a	n/a	340	n/a	n/a
ClearEvent			n/a	n/a	n/a	364	364	364
GetEvent			n/a	n/a	n/a	264	264	264
WaitEvent	<default>		n/a	n/a	n/a	776	776	1072
	fp	11	n/a	n/a	n/a	820	820	1172

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	436	n/a	n/a
GetAlarmBase			404	404	404	404	404	404
GetAlarm			372	372	372	372	372	372
SetRelAlarm			460	460	460	460	460	460
SetAbsAlarm			484	484	484	484	484	484
CancelAlarm			352	352	352	352	352	352
InitCounter			440	440	440	440	440	440
GetCounterValue			416	416	416	416	416	416
osek_tick_alarm	<default>		300	300	300	300	300	300
	KL	2	68	68	68	68	68	68
osek_incr_counter			60	60	60	60	60	60
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			432	432	432	432	432	432
ShutdownOS	NoHook	12	124	124	124	124	124	124
	Hook	13	148	148	148	148	148	148
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			92	92	92	92	92	92
StopCOM			68	68	68	68	68	68
ReadFlag			44	44	44	44	44	44
ResetFlag			52	52	52	52	52	52
ReceiveMessage	CCCA	14	348	348	348	516	516	516
	CCCB	15	516	516	516	516	516	516
GetMessageResource			160	160	160	160	160	160
ReleaseMessageResource			160	160	160	160	160	160
GetMessageStatus			184	184	184	184	184	184
SendMessage	SW CCCA	1, 14	428	428	428	628	628	628
	SW CCCB	1, 15	604	604	604	628	628	628
	NS CCCA	14	428	428	428	628	628	628
	NS CCCB	15	604	604	604	628	628	628
	KL CCCA	2, 14	296	296	296	496	496	496
	KL CCCB	2, 15	472	472	472	496	496	496
main_dispatch	NoHook	12	248	248	308	248	248	308
	Hook	13	300	300	360	300	300	360
sub_dispatch	B1LF	19	36	36	36	36	36	36
	B1HI	20	132	132	132	132	132	132
	B1HF	21	140	140	140	140	140	140

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	B2LI	22	n/a	80	124	n/a	80	124
	B2LF	23	n/a	84	128	n/a	84	128
	B2HI	24	n/a	180	272	n/a	180	272
	B2HF	25	n/a	188	280	n/a	188	280
	E1HI	26	n/a	n/a	n/a	632	632	720
	E1HF	27	n/a	n/a	n/a	640	640	728
	E2HI	28	n/a	n/a	n/a	n/a	n/a	720
	E2HF	29	n/a	n/a	n/a	n/a	n/a	728
ErrorHook support		16	312	312	312	312	312	312
	ServiceID	17	320	320	320	320	320	320
	Parameters	18	344	344	344	344	344	344
validity_checks		3	56	56	56	56	56	56
Timing_dispatch		4	160	160	160	160	160	160
Timing_termination		4	172	172	172	172	172	172
ActivateTaskset	SW	1	612	680	736	624	704	808
	NS		700	780	840	712	812	912
	KL	2	420	512	544	436	540	604
ChainTaskset	SWL	1, 8	668	732	788	676	752	848
	SWH	1, 9	756	848	904	764	868	988
	NSL	8	784	864	936	792	884	988
	NSH	9	876	960	1032	884	980	1096
GetTasksetRef			228	228	228	228	228	228
MergeTaskset			504	504	504	504	504	504
AssignTaskset			256	256	256	256	256	256
RemoveTaskset			504	504	504	504	504	504
TestSubTaskset			536	536	536	536	536	536
TestEquivalentTaskset			532	532	532	532	532	532
TickSchedule	SW	1	596	512	512	512	512	512
	NS		684	624	624	624	624	624
	KL	2	440	356	356	356	356	356
AdvanceSchedule	SW	1	608	520	520	520	520	520
	NS		696	632	632	632	632	632
	KL	2	464	376	376	376	376	376
StartSchedule			492	492	492	492	492	492
StopSchedule			408	408	408	408	408	408
GetScheduleStatus			464	464	464	464	464	464
GetScheduleValue			376	376	376	376	376	376

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleNext			140	140	140	140	140	140
SetScheduleNext			264	264	264	264	264	264
GetArrivalpointDelay			200	200	200	200	200	200
SetArrivalpointDelay			228	228	228	228	228	228
GetArrivalpointTasksetRef			196	196	196	196	196	196
GetArrivalpointNext			200	200	200	200	200	200
SetArrivalpointNext			284	284	284	284	284	284
TestArrivalpointWritable			232	232	232	232	232	232
GetExecutionTime			332	332	332	332	332	332
GetLargestExecutionTime			184	184	184	184	184	184
ResetLargestExecutionTime			168	168	168	168	168	168
GetStackOffset			20	20	20	20	20	20
Stack manipulation			84	84	84	84	84	84
Interrupt support			296	296	296	296	296	296
Interrupt support			344	344	344	344	344	344

Notes

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

Number	Note
	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

4.2.4 Reserved Hardware Resources

4.3 Performance

4.3.1 Execution Times for RTA-OSEK API Calls

The following tables give the execution time (in CPU cycles) for each API call. (Note that: (1) the OSEK COM class was set to CCCA for systems without events and to CCCB for systems with events; (2) `ShutdownOS()` enters an infinite loop; the execution time for `ShutdownOS()` reported below is the time up to the point at which `ShutdownOS()` calls `ShutdownHook()`).

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	790	973	1201	811	910	1225
	NS	891	1234	1539	867	1230	1497

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	KL	275	434	946	287	542	920
TerminateTask	LExt	0	0	0	0	0	0
	H	1558	1602	1581	1560	1554	1608
ChainTask	SWL	1788	2112	2598	2092	2221	2827
	SWH	2751	3145	3681	3212	3436	3748
	NSL	1999	2440	3004	2327	2792	3233
	NSH	2953	3271	3847	3164	3368	4004
Schedule	SW	830	798	836	1040	1004	824
GetTaskID		119	159	157	157	118	171
GetTaskState		866	872	863	914	914	917
EnableAllInterrupts		247	173	196	169	231	169
DisableAllInterrupts		306	353	282	311	308	314
ResumeAllInterrupts		256	247	256	285	295	295
SuspendAllInterrupts		464	383	425	449	440	446
ResumeOSInterrupts		175	175	201	211	175	175
SuspendOSInterrupts		224	224	251	260	231	260
GetResource	Task	254	245	300	230	299	233
	Combined	555	591	561	561	754	564
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	975	942	933	1005	999	975
	Combined	1182	1182	1369	1143	1380	1143
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	1079	922	976
	NS	n/a	n/a	n/a	982	940	1074
	KL	n/a	n/a	n/a	397	370	459
ClearEvent		n/a	n/a	n/a	600	575	563
GetEvent		n/a	n/a	n/a	106	142	109
WaitEvent	<default>	n/a	n/a	n/a	3531	3699	3795
	fp	n/a	n/a	n/a	3876	3918	4146
GetAlarmBase		941	866	845	836	902	877
GetAlarm		966	997	986	998	1028	992
SetRelAlarm		883	918	1101	1131	996	972
SetAbsAlarm		1026	987	996	1047	1020	1059
CancelAlarm		627	627	627	627	627	627
InitCounter		535	535	503	503	499	502
GetCounterValue		758	749	755	755	755	755

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
osek_tick_alarm	<default>	613	856	853	871	625	829
	KL	228	261	311	269	188	219
osek_incr_counter		157	115	115	115	115	115
GetActiveApplicationMode		81	81	45	45	45	45
StartOS		5070	5040	5067	5100	5007	5013
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	454	445	446	419	455	422
InitCOM		10	49	42	49	46	41
CloseCOM		1	1	2	1	27	1
StartCOM		121	119	121	407	451	409
StopCOM		88	51	52	51	51	52
ReadFlag		n/a	n/a	n/a	112	114	114
ResetFlag		n/a	n/a	n/a	83	99	83
ReceiveMessage		760	817	793	1680	1638	1914
GetMessageResource		n/a	n/a	n/a	788	863	828
ReleaseMessageResource		n/a	n/a	n/a	1139	1137	1137
GetMessageStatus		n/a	n/a	n/a	240	201	306
SendMessage	SW	1404	1623	1857	2470	2650	2835
	NS	1801	1836	2223	3139	3369	3141
	KL	686	878	1061	1954	2137	2203
ActivateTaskset	SW	850	1255	1531	937	1546	2059
	NS	655	1180	1675	712	1228	1788
	KL	84	531	912	113	572	1105
	SW2	726	1101	1491	720	1068	1716
	NS2	562	859	1362	478	877	1350
	KL2	84	492	831	113	458	868
ChainTaskset	SWL	1872	2286	3069	2023	2464	3355
	SWH	2805	3223	3865	2990	3377	4163
	NSL	1840	2245	2983	1940	2408	3200
	NSH	2722	3223	3859	2996	3401	4118
GetTasksetRef		142	176	134	142	142	142
MergeTaskset		738	729	738	731	738	726
AssignTaskset		97	139	106	139	97	97
RemoveTaskset		704	728	732	719	672	692
TestSubTaskset		753	719	734	738	762	771
TestEquivalentTaskset		702	753	711	753	714	711

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
TickSchedule	SW	1447	1969	2347	1540	2008	2536
	NS	1112	1712	2061	1387	1789	2197
	KL	566	1220	1517	809	1274	1667
	SW2	1017	1587	1923	1206	1551	1962
	NS2	759	1429	1754	1037	1382	1793
	KL2	413	1025	1322	605	950	1361
AdvanceSchedule	SW	1249	1816	2203	1453	1873	2206
	NS	1096	1618	1915	1213	1678	2071
	KL	517	1049	1382	674	1160	1489
	SW2	951	1419	1755	1038	1383	1794
	NS2	779	1298	1634	917	1262	1898
	KL2	365	899	1235	518	863	1291
StartSchedule		736	700	727	736	694	736
StopSchedule		668	642	684	659	642	668
GetScheduleStatus		1015	1042	1042	1030	1072	1015
GetScheduleValue		798	828	822	795	795	798
GetScheduleNext		104	126	126	104	137	104
SetScheduleNext		123	101	101	123	132	123
GetArrivalpointDelay		138	138	138	143	138	96
SetArrivalpointDelay		97	75	75	97	106	139
GetArrivalpointTasksetRef		38	47	52	40	94	38
GetArrivalpointNext		104	98	138	105	74	74
SetArrivalpointNext		97	106	75	97	97	139
TestArrivalpointWritable		64	127	151	81	149	97
GetExecutionTime		39	10	6	86	48	95
GetLargestExecutionTime		51	128	83	83	89	51
ResetLargestExecutionTime		124	82	88	88	82	124
GetStackOffset		40	92	4	98	66	98

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	790	1012	1285	769	868	1180
	NS	855	1218	1603	900	1137	1509
	KL	248	683	869	359	514	764
TerminateTask	LExt	0	0	0	0	0	0
	H	2529	2529	2595	2568	2536	2622
ChainTask	SWL	3103	3391	3867	3458	3574	4256
	SWH	3903	4209	5025	4645	4654	5044
	NSL	3379	3823	4357	3728	3911	4574
	NSH	4252	4552	5089	4538	4688	5234
Schedule	SW	773	770	815	828	782	1041
GetTaskID		158	157	160	161	158	160
GetTaskState		881	884	884	914	908	902
EnableAllInterrupts		181	169	195	280	166	205
DisableAllInterrupts		283	308	308	314	324	356
ResumeAllInterrupts		306	273	283	256	256	256
SuspendAllInterrupts		401	419	413	435	425	426
ResumeOSInterrupts		211	211	175	175	201	175
SuspendOSInterrupts		257	260	228	221	251	221
GetResource	Task	230	230	266	245	267	248
	Combined	564	564	529	555	561	558
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	978	948	1023	979	1038	943
	Combined	1143	1281	1185	1182	1225	1182
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	1096	904	964
	NS	n/a	n/a	n/a	949	982	1170
	KL	n/a	n/a	n/a	358	397	444
ClearEvent		n/a	n/a	n/a	566	609	597
GetEvent		n/a	n/a	n/a	142	142	142
WaitEvent	<default>	n/a	n/a	n/a	4422	4506	4743
	fp	n/a	n/a	n/a	4731	4780	5016
GetAlarmBase		887	899	869	920	878	908
GetAlarm		989	992	1034	998	998	995

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		807	978	1053	1065	804	1038
SetAbsAlarm		1032	1059	1015	1009	1005	991
CancelAlarm		627	627	627	627	627	627
InitCounter		499	499	535	535	535	535
GetCounterValue		746	755	764	773	755	770
osek_tick_alarm	<default>	751	823	790	721	871	790
	KL	227	258	313	214	311	301
osek_incr_counter		115	154	154	154	134	154
GetActiveApplicationMode		45	45	84	84	81	84
StartOS		11268	11310	11274	11307	11322	11292
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	467	452	461	488	439	476
InitCOM		6	6	6	10	39	10
CloseCOM		29	29	6	71	26	33
StartCOM		157	157	125	445	409	443
StopCOM		78	78	88	88	88	87
ReadFlag		n/a	n/a	n/a	111	117	111
ResetFlag		n/a	n/a	n/a	76	86	76
ReceiveMessage		796	820	754	1675	1897	2047
GetMessageResource		n/a	n/a	n/a	824	827	785
ReleaseMessageResource		n/a	n/a	n/a	1175	1179	1166
GetMessageStatus		n/a	n/a	n/a	239	240	247
SendMessage	SW	1398	1623	1867	2469	2553	2845
	NS	1539	2001	2094	3102	3300	3205
	KL	569	797	1028	1885	2047	2206
ActivateTaskset	SW	985	1105	1669	1006	1456	1966
	NS	649	1105	1695	683	1207	1720
	KL	123	531	963	152	581	1081
	SW2	690	1101	1491	720	1068	1479
	NS2	448	901	1362	478	877	1350
	KL2	84	492	831	152	500	868
ChainTaskset	SWL	3175	3637	4392	3443	3802	4685
	SWH	4063	4549	5235	4366	4843	5566
	NSL	3028	3514	4234	3287	3695	4589
	NSH	4024	4468	5203	4244	4652	5390
GetTasksetRef		129	165	142	104	129	104

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		725	728	719	738	725	714
AssignTaskset		105	105	139	136	105	136
RemoveTaskset		729	729	710	741	729	729
TestSubTaskset		774	774	774	719	774	719
TestEquivalentTaskset		716	692	717	756	692	756
TickSchedule	SW	1216	1996	2377	1285	1870	2365
	NS	1118	1753	2095	1366	1816	2184
	KL	524	1187	1478	824	1244	1670
	SW2	1017	1587	1920	1210	1551	1966
	NS2	759	1426	1762	1045	1382	1796
	KL2	413	986	1322	605	950	1361
AdvanceSchedule	SW	1108	1768	2203	1450	1885	2101
	NS	1100	1613	1945	1180	1655	2041
	KL	479	1037	1352	704	1139	1502
	SW2	951	1419	1752	1038	1383	1794
	NS2	779	1328	1637	917	1292	1673
	KL2	365	899	1235	518	1019	1274
StartSchedule		736	736	784	721	736	721
StopSchedule		678	678	656	684	678	684
GetScheduleStatus		1045	1048	1012	1039	1042	1084
GetScheduleValue		762	771	792	792	789	795
GetScheduleNext		126	126	104	126	126	126
SetScheduleNext		123	123	123	101	123	101
GetArrivalpointDelay		82	104	107	138	82	96
SetArrivalpointDelay		139	97	139	75	139	105
GetArrivalpointTasksetRef		15	48	4	52	15	52
GetArrivalpointNext		96	96	105	138	138	138
SetArrivalpointNext		141	144	139	75	108	75
TestArrivalpointWritable		108	108	77	151	136	160
GetExecutionTime		761	761	820	770	761	770
GetLargestExecutionTime		214	172	214	214	175	205
ResetLargestExecutionTime		106	106	123	110	148	110
GetStackOffset		42	42	33	4	42	4

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	1771	1993	2182	1750	1846	2110
	NS	2637	2859	3133	2562	2685	3159
	KL	1264	1447	1930	1483	1384	1852
TerminateTask	LExt	2679	2670	2676	2655	2610	2652
	H	3420	3408	3453	3522	3369	3393
ChainTask	SWL	4969	5587	6003	5506	5768	5618
	SWH	5895	6099	6805	6349	6556	7054
	NSL	5665	6067	6613	5936	6194	6848
	NSH	6286	6595	7291	7079	6761	7487
Schedule	SW	1366	1233	1552	1579	1676	1820
GetTaskID		222	216	219	177	258	213
GetTaskState		2266	2244	2311	2164	2191	2197
EnableAllInterrupts		222	225	237	249	280	229
DisableAllInterrupts		352	355	423	363	383	377
ResumeAllInterrupts		546	585	488	503	517	478
SuspendAllInterrupts		475	436	484	514	496	478
ResumeOSInterrupts		364	364	328	328	328	328
SuspendOSInterrupts		286	286	250	250	250	247
GetResource	Task	3592	3589	2536	3862	3845	2765
	Combined	1822	1816	2001	2139	2160	2739
	CLEx	2700	2709	2709	2868	2922	2964
ReleaseResource	Task	2637	2628	2484	2802	2781	2745
	Combined	2700	2715	2583	2685	2613	2901
	CLEx	2445	2304	2355	2598	2640	2640
SetEvent	SW	n/a	n/a	n/a	1888	2297	2270
	NS	n/a	n/a	n/a	2721	2721	2796
	KL	n/a	n/a	n/a	1894	1924	1942
ClearEvent		n/a	n/a	n/a	1342	1319	1325
GetEvent		n/a	n/a	n/a	1515	1453	1516
WaitEvent	<default>	n/a	n/a	n/a	5508	5517	6255
	fp	n/a	n/a	n/a	5703	5658	5946
GetAlarmBase		1861	1902	1939	1873	1834	1900
GetAlarm		1907	1914	1924	1931	1918	1954

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		2205	2139	2224	2137	2185	2185
SetAbsAlarm		2068	2043	2049	2022	2112	2073
CancelAlarm		1332	1335	1332	1332	1332	1332
InitCounter		2154	2289	2181	2034	2034	2370
GetCounterValue		1774	1710	1666	1690	1666	1783
osek_tick_alarm	<default>	1219	1179	1258	1255	1120	1210
	KL	267	267	300	300	282	308
osek_incr_counter		175	175	116	116	116	116
GetActiveApplicationMode		84	84	81	81	45	45
StartOS		11784	11766	11775	11772	11742	11793
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	524	524	487	478	503	485
InitCOM		5	1	49	6	6	6
CloseCOM		5	1	1	6	2	2
StartCOM		295	273	288	583	570	570
StopCOM		115	130	79	115	83	83
ReadFlag		n/a	n/a	n/a	159	198	198
ResetFlag		n/a	n/a	n/a	158	56	98
ReceiveMessage		1507	1515	1564	2725	2746	2749
GetMessageResource		n/a	n/a	n/a	3396	3100	3304
ReleaseMessageResource		n/a	n/a	n/a	2839	2842	2839
GetMessageStatus		n/a	n/a	n/a	824	854	857
SendMessage	SW	3028	3195	3372	3936	4002	4356
	NS	3723	4542	4845	5073	5631	6135
	KL	2410	3126	3400	3967	4153	4576
ActivateTaskset	SW	2098	2647	2983	2014	2407	3238
	NS	2427	3081	3366	2353	3045	3525
	KL	1213	2014	2038	1015	1684	1909
	SW2	1374	1917	2250	1404	2058	2340
	NS2	1632	2226	2514	1656	2250	2667
	KL2	823	1438	1780	877	1345	1840
ChainTaskset	SWL	5296	5896	6609	5596	6025	6901
	SWH	6036	6657	7335	6485	6962	7721
	NSL	5758	6376	6982	5909	6461	7487
	NSH	6289	6829	7675	6818	6977	7988
GetTasksetRef		1149	1152	1305	1111	1158	1218

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		1493	1568	1667	1568	1582	1582
AssignTaskset		602	599	571	626	577	577
RemoveTaskset		1540	1567	1580	1577	1595	1595
TestSubTaskset		1709	1673	1637	1666	1661	1661
TestEquivalentTaskset		1646	1556	1642	1604	1531	1531
TickSchedule	SW	2215	3838	3799	2917	4030	3982
	NS	2515	3781	4114	3177	4159	4339
	KL	1537	2707	3043	2116	2866	3223
	SW2	1518	2802	3144	2248	3094	3381
	NS2	1751	3035	3587	2891	3949	3479
	KL2	991	2278	2659	1759	2329	2677
AdvanceSchedule	SW	2221	3343	3664	2569	3391	3871
	NS	2488	3604	4036	3037	3719	4175
	KL	1504	2683	2977	2104	2752	3238
	SW2	1467	2757	3399	2157	2769	3159
	NS2	1700	3455	3575	2339	3281	3329
	KL2	973	2338	2563	1894	2134	2626
StartSchedule		1168	1201	1294	1156	1168	1168
StopSchedule		879	876	915	921	915	915
GetScheduleStatus		1519	1489	1453	1507	1459	1453
GetScheduleValue		1119	1062	1102	1083	1068	1095
GetScheduleNext		356	356	390	420	390	390
SetScheduleNext		431	398	358	409	377	377
GetArrivalpointDelay		524	524	578	545	517	547
SetArrivalpointDelay		653	653	586	565	593	605
GetArrivalpointTasksetRef		482	482	449	479	421	418
GetArrivalpointNext		442	442	470	494	479	470
SetArrivalpointNext		701	701	676	682	676	673
TestArrivalpointWritable		467	467	476	572	497	455
GetExecutionTime		997	997	991	1024	979	979
GetLargestExecutionTime		1236	1236	1233	1203	1287	1254
ResetLargestExecutionTime		1176	1176	1098	1123	1089	1086
GetStackOffset		51	93	47	32	81	42

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	323	323	323	323	323	323
	Cat 2	732	729	729	729	729	732

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	323	323	323	323	323	323
	Cat 2	1411	1411	1411	1411	1411	1411

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	323	323	323	323	323	323
	Cat 2	1411	1411	1414	1411	1411	1411

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.

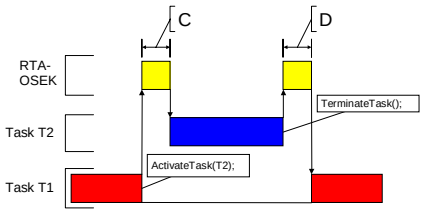

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

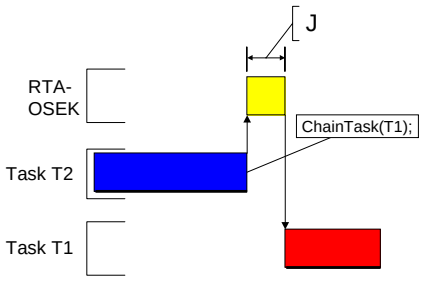

Figure 2: Task Chaining

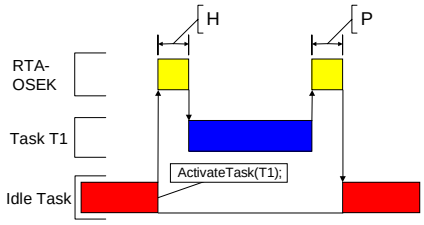

Figure 3: Task Activation from Idle Task

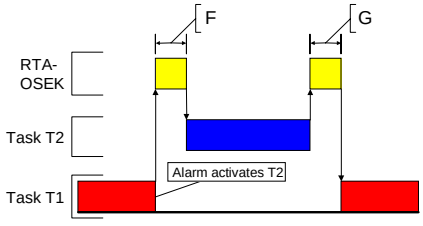

Figure 4: Task Activation from an Alarm

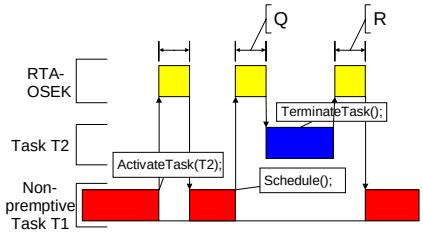

Figure 5: Non-Premptive Task Calls Schedule()

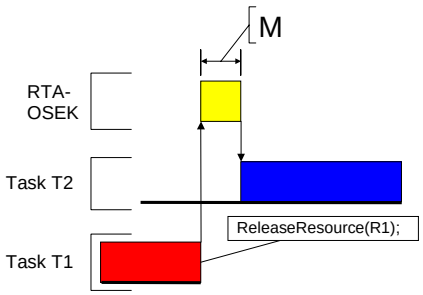

Figure 6: Blocked Task Activated by ReleaseResource()

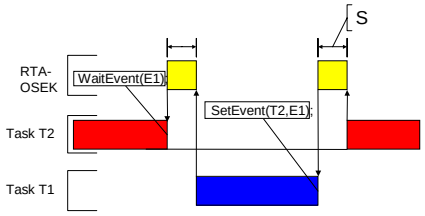

Figure 7: Waiting Task Activated by SetEvent()

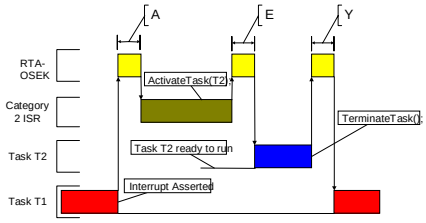

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	779	1046	1235	779	1046	1262
Figure 1: D	Heavy, Basic/Extended	1574	1820	2003	1922	1874	2060
ChainTask	Light, Basic	1095	1428	1914	1077	1394	1968
Figure 2: J	Heavy, Basic/Extended	3476	4061	4703	3800	4155	4811
Pre-emption	Light, Basic	1339	1645	2179	1363	1675	2233
Figure 1: C	Heavy, Basic/Extended	2140	2359	2863	2345	2468	3056
From idle task	Light, Basic	1340	1646	2183	1364	1649	2237
Figure 3: H	Heavy, Basic/Extended	2108	2333	2864	2343	2439	3027
Triggered by alarm	Light, Basic	1971	2487	2905	2160	2412	3000
Figure 4: F	Heavy, Basic/Extended	3039	3246	3786	3310	3397	3961
Schedule	Light, Basic	1224	1308	1662	1221	1308	1665
Figure 5: Q	Heavy, Basic/Extended	2229	2190	2628	2399	2407	2794
Release resource	Light, Basic	1416	1500	2094	1413	1764	2142
Figure 6: M	Heavy, Basic/Extended	2493	2454	2778	2659	2659	3004
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	4485	4494	5197

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
From category 2 ISR	Light, Basic	1029	1113	1413	1026	1113	1416
Figure 8: E	Heavy, Basic/Extended	1800	1797	2097	2008	2008	2311

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	1808	2132	2141	1868	1954	2168
Figure 1: D	Heavy, Basic/Extended	2549	2840	2891	2777	2722	2900
ChainTask	Light, Basic	2393	2729	3237	2411	2763	3257
Figure 2: J	Heavy, Basic/Extended	5823	6417	6873	6330	6281	7185
Pre-emption	Light, Basic	1735	2011	2539	1846	2011	2593
Figure 1: C	Heavy, Basic/Extended	2449	2677	3205	2801	2816	3398
From idle task	Light, Basic	1739	2150	2543	1862	2165	2597
Figure 3: H	Heavy, Basic/Extended	2450	2831	3206	2805	2970	3399
Triggered by alarm	Light, Basic	2517	2653	3285	2655	2908	3235
Figure 4: F	Heavy, Basic/Extended	3366	3612	4176	3718	3739	4402
Schedule	Light, Basic	1620	1704	2020	1692	1770	2217
Figure 5: Q	Heavy, Basic/Extended	2562	2568	2926	2773	2857	3166
Release resource	Light, Basic	1854	1896	2304	2130	2088	2235
Figure 6: M	Heavy, Basic/Extended	2826	2838	3084	3121	3109	3412
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	4704	4653	5499
From category 2 ISR	Light, Basic	2460	2511	2805	2499	2511	2802
Figure 8: E	Heavy, Basic/Extended	3174	3177	3471	3442	3418	3715

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	2642	2774	2966	2642	2894	3008
Figure 1: D	Heavy, Basic/Extended	3362	3458	3716	3530	3596	3806
ChainTask	Light, Basic	3899	4241	4832	3969	4253	4781
Figure 2: J	Heavy, Basic/Extended	8098	8492	9160	8237	8595	9296
Pre-emption	Light, Basic	2581	2851	3379	2605	2851	3412
Figure 1: C	Heavy, Basic/Extended	3295	3517	4045	3566	3782	4214
From idle task	Light, Basic	2581	2851	3382	2609	3085	3560
Figure 3: H	Heavy, Basic/Extended	3295	3517	4048	3566	4094	4289
Triggered by alarm	Light, Basic	3513	3943	4689	3682	4123	4341
Figure 4: F	Heavy, Basic/Extended	4632	4782	5430	4939	5242	5644
Schedule	Light, Basic	1932	2316	2712	2196	2394	2832
Figure 5: Q	Heavy, Basic/Extended	2952	2982	3382	3259	3457	3619
Release resource	Light, Basic	3210	3144	3102	3426	3228	3678
Figure 6: M	Heavy, Basic/Extended	4002	3996	4294	4417	4615	4729
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	5769	5784	6579
From category 2 ISR	Light, Basic	2686	2737	3070	2689	2737	3028
Figure 8: E	Heavy, Basic/Extended	3430	3433	3805	3650	3644	3935

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	176	176	192	176	176	192
BCC1 lightweight, floating-point	192	192	208	192	192	208
BCC1 heavyweight, integer	448	448	464	448	448	464
BCC1 heavyweight, floating-point	448	448	464	448	448	464
BCC2 lightweight, integer	n/a	192	208	n/a	192	208
BCC2 lightweight, floating-point	n/a	192	208	n/a	192	208
BCC2 heavyweight, integer	n/a	448	464	n/a	448	464
BCC2 heavyweight, floating-point	n/a	448	464	n/a	448	464
ECC1 heavyweight, integer	n/a	n/a	n/a	464	464	480
ECC1 heavyweight, floating-point	n/a	n/a	n/a	464	464	480
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	480
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	480
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	192	192	192	192	192	192
BCC1 lightweight, floating-point	208	208	208	208	208	208
BCC1 heavyweight, integer	464	464	464	464	464	464
BCC1 heavyweight, floating-point	464	464	464	464	464	464
BCC2 lightweight, integer	n/a	208	208	n/a	208	208
BCC2 lightweight, floating-point	n/a	208	208	n/a	208	208
BCC2 heavyweight, integer	n/a	464	464	n/a	464	464
BCC2 heavyweight, floating-point	n/a	464	464	n/a	464	464
ECC1 heavyweight, integer	n/a	n/a	n/a	480	480	480
ECC1 heavyweight, floating-point	n/a	n/a	n/a	480	480	480
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	480
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	480

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	208	208	208	208	208	208
BCC1 lightweight, floating-point	224	224	224	224	224	224
BCC1 heavyweight, integer	480	480	480	480	480	480
BCC1 heavyweight, floating-point	480	480	480	480	480	480
BCC2 lightweight, integer	n/a	224	224	n/a	224	224
BCC2 lightweight, floating-point	n/a	224	224	n/a	224	224
BCC2 heavyweight, integer	n/a	480	480	n/a	480	480
BCC2 heavyweight, floating-point	n/a	480	480	n/a	480	480
ECC1 heavyweight, integer	n/a	n/a	n/a	496	496	496
ECC1 heavyweight, floating-point	n/a	n/a	n/a	496	496	496
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	496
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	496
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	208	208	208	208	208	208
BCC1 lightweight, floating-point	224	224	224	224	224	224
BCC1 heavyweight, integer	480	480	480	480	480	480
BCC1 heavyweight, floating-point	480	480	480	480	480	480
BCC2 lightweight, integer	n/a	224	224	n/a	224	224
BCC2 lightweight, floating-point	n/a	224	224	n/a	224	224
BCC2 heavyweight, integer	n/a	480	480	n/a	480	480
BCC2 heavyweight, floating-point	n/a	480	480	n/a	480	480
ECC1 heavyweight, integer	n/a	n/a	n/a	496	496	496
ECC1 heavyweight, floating-point	n/a	n/a	n/a	496	496	496
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	496
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	496

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	208	208	208	208	208	208
BCC1 lightweight, floating-point	224	224	224	224	224	224
BCC1 heavyweight, integer	480	480	480	480	480	480
BCC1 heavyweight, floating-point	480	480	480	480	480	480
BCC2 lightweight, integer	n/a	224	224	n/a	224	224
BCC2 lightweight, floating-point	n/a	224	224	n/a	224	224
BCC2 heavyweight, integer	n/a	480	480	n/a	480	480
BCC2 heavyweight, floating-point	n/a	480	480	n/a	480	480
ECC1 heavyweight, integer	n/a	n/a	n/a	496	496	496
ECC1 heavyweight, floating-point	n/a	n/a	n/a	496	496	496
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	496
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	496
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	208	208	208	208	208	208
BCC1 lightweight, floating-point	224	224	224	224	224	224
BCC1 heavyweight, integer	480	480	480	480	480	480
BCC1 heavyweight, floating-point	480	480	480	480	480	480
BCC2 lightweight, integer	n/a	224	224	n/a	224	224
BCC2 lightweight, floating-point	n/a	224	224	n/a	224	224
BCC2 heavyweight, integer	n/a	480	480	n/a	480	480
BCC2 heavyweight, floating-point	n/a	480	480	n/a	480	480
ECC1 heavyweight, integer	n/a	n/a	n/a	496	496	496
ECC1 heavyweight, floating-point	n/a	n/a	n/a	496	496	496
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	496
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	496

Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found on the ETAS Group website www.etasgroup.com.