RTA-OSEK

Binding Manual: V850E/GreenHills

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: RM00056-002

Disclaimer

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

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

Trademarks

RTA-OSEK 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 5
1.1 Who Should Read this Guide? 5
1.2 Conventions 5
2 Toolchain Issues 7
2.1 Compiler 7
2.2 Assembler 8
2.3 Linker/Locator 8
2.4 Debugger 8
3 Target Hardware Issues 11
3.1 Interrupts 11
3.1.1 Interrupt Levels 11
3.1.2 Interrupt Vectors 11
3.1.3 Category 1 Handlers 11
3.1.4 Category 2 Handlers 12

3.1.5 Vector Table Issues 12
3.1.6 Traps 12
3.1.7 Default Interrupt 13
3.1.8 Flash Security ID 13
3.2 Register Settings 14
3.3 Stack Usage 14
3.3.1 Number of Stacks 14
3.3.2 Stack Usage within API Calls 14
4 Parameters of Implementation 17
4.1 Functionality 17
4.2 Hardware Resources 18
4.2.1 ROM and RAM Overheads 18
4.2.2 ROM and RAM for OSEK OS Objects 19
4.2.3 Size of Linkable Modules 24
4.2.4 Reserved Hardware Resources 37
4.3 Performance 37
4.3.1 Execution Times for RTA-OSEK API Calls 37
4.3.2 OS Start-up Time 46
4.3.3 Interrupt Latencies 46
4.3.4 Task Switching Times 47
4.4 Configuration of Run-time Context 51

1 About this Guide

This guide provides port specific information for the V850E/GreenHills 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.

2.1 Compiler

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

Vendor	Green Hills Software, Inc.
Compiler	ccv850e
Version	4.0.7A

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

Option	Description
`-O`	Use optimizations
`-noobj`	Turn off binary code generation
`-r21has65535`	r20 always contains 255, r21 always contains 65535

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

Option	Description
`-O`	Use optimizations
`-noobj`	Turn off binary code generation
`-r21has65535`	r20 always contains 255, r21 always contains 65535

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

Option	Description
`-g`	Debug
`-ZDA`	Use ZDA
`-sda`	Use SDA

2.2 Assembler

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

Vendor	Green Hills Software, Inc.
Assembler	ccv850e
Version	4.0.7A

The assembly file that RTA-OSEK generates from your OIL configuration file is called `osgen.850`. This file defines configuration parameters for the RTA-OSEK Component when running your application.

2.3 Linker/Locator

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. This is only accessed during StartOS().
`os_intvec`	ROM	Vector table (if generated by RTA-OSEK)
`os_text`	ROM	RTA-OSEK code section.
`os_pir`	RAM	RTA-OSEK initialized data. Must be initialized during C-startup.
`os_pur`	RAM	RTA-OSEK uninitialized data. Must be zeroed during C-startup.
`os_pirf`	RAM	RTA-OSEK uninitialized data. Must be initialized during C-startup.

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

C Library Functions	Description
`setjmp`	Saves the current state of the program. (Used for heavyweight task termination.)
`longjmp`	Restores stack environment and execution locale. (Used for heavyweight task termination.)

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
iSYSTEM winIDEA

To resolve the ORTI data in the debugger the file `osekdefs.c` must be compiled with DWARF debug information generated in the object file. The compiler options `-g -dual_debug -dwarf` generate the required debug information.

To resolve the `CURRENTSERVICE` and `CURRENTAPPMODE` values additional ROM pointers are added into `osekdefs.c`. If these values are not required then the macro `OS_NO_ORTI` should be defined when the file is compiled, also no debugger should be selected in the RTA-OSEK GUI.

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 *appropriate NEC manuals*.

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

IPL Value	ID Bit PSW Register	Description
0	0	User level
1	1	Category 1 and 2 maskable interrupts.
2	1	NMIs, exceptions and TRAPs (can preempt maskable interrupts).

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:

Vectors	Restriction
0x0010 to 0x0070	Category 1, IPL 2
0x0080 to maximum vector for CPU variant	Category 1 or 2, IPL 1

The valid base addresses for the vector table are:

0x0010	The vector table must start just above the reset vector.

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 Green Hills Software 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

The number of vectors available depends upon the specific V850 chip variant selected in RTA-OSEK. The variants directly supported are FE2, FF2, FG2, FG3, FJ2, FG3, PH2, PH3, RS1, SG2, SG3, SJ2 and SJ3. These are in addition to the 'Generic V850E' variant. Further variants can be supported by contacting LiveDevices.

When RTA-OSEK generates an interrupt vector table for the V850, it only emits data for addresses 0x0010 up to the highest declared interrupt. This allows RTA-OSEK to cope efficiently with chip variants with differently sized vector tables.

Important: When you configure your application with the RTA-OSEK GUI you can choose whether or not a vector table is generated within `osgen.850`. It is not recommended that you attempt to do this for the GHSV850E port, because RTA-OSEK generates optimized code that fits into the 16 bytes allocated to each interrupt source. This code must not be modified

3.1.6 Traps

If a trap instruction is used, a Category 1 interrupt handler can be set up to service the exception. However, RTA-OSEK does not preserve the EP (exception in progress) bit in the PSW if an API call is made that manipulates the IPL. If such calls are used, the EP bit must be set back to 1 prior to leaving the interrupt handler as shown in the sample code below.

```
asm void set_PSW(ByteType m) {
%reg m ;
 ldsr m, PSW ;
%error
 Macro has not expanded
}

asm ByteType get_PSW(void) {
 stsr PSW,r10;
}

__interrupt void sync_isr(void)
{
 register ByteType psw_val = get_PSW();

 DisableAllInterrupts();
 ...
 EnableAllInterrupts();

 set_PSW(psw_val);
}
```

Note that in reality an exception handler never needs to make such calls, because it is already executing at the highest IPL and it is illegal for it to lower the interrupt priority. In this case, no special processing will be needed.

3.1.7 Default Interrupt

The 'default interrupt' is intended to be used to catch all unexpected interrupts. All unused interrupts have their interrupt vectors directed to the named routine that you specify. This routine must correctly handle the interrupt context, in the same way as a Category 1 ISR. The Green Hills C compiler can generate appropriate interrupt handling code using the `__interrupt` function qualifier.

Because RTA-OSEK only emits interrupt vectors for addresses 0x0010 up to the highest declared interrupt, it will only fill unused vectors with the default interrupt up to the highest declared interrupt. To fill the entire vector table for your chip variant, create a dummy Category 1 interrupt and place it on the highest vector used by the chip. The default interrupt will then be used to fill all unused vectors below this.

3.1.8 Flash Security ID

To protect the contents of internal ROM memory some V850 variants such as the FE2 and FF2 support a 10 byte security number located at memory address 0x70. This address falls within the interrupt vector table range. If the user wishes to enter a 10 byte security number at this address the `OS_SECURITY_ID` macro in `osgen.850` can be used. For example

assembling `osgen.850` with the command line option `-DOS_SECURITY_ID=0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x99, 0xAA` insets the security number 0x1122334455667788 99AA at address 0x70.

3.2 Register Settings

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

Register	Value
r20	255
r21	65535

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

Register	Notes
PSW.ID	Processor status word interrupt enable bit

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): 36

Timing

API max usage (bytes): 36

Extended

API max usage (bytes): 76

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.

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	**"V850E/PH2"**
Clock speed (MHz)	64
Code memory	On-chip FLASH ROM
Read-only data memory	On-chip FLASH ROM
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	64	64	64	64	64	64
Maximum number of not suspended tasks	64	64	64	64	64	64
Maximum number of priorities	64	64	64	64	64	64
Number of tasks per priority (for BCC2 and ECC2)	n/a	64	64	n/a	64	64
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	40	40	40	40	40	40
	ROM	202	202	202	202	202	202
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	52	52	52	52	52	52
	ROM	250	250	250	250	250	250
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	80	80	80	80	80	80
	ROM	306	306	306	306	306	306
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	44	44	44	44	44	44
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	48	48	48	48	48	48
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	52	60	n/a	52	60
ECC1, Integer task	RAM	n/a	n/a	n/a	84	84	84
	ROM	n/a	n/a	n/a	68	68	68
ECC1, floating-point task	RAM	n/a	n/a	n/a	88	88	88
	ROM	n/a	n/a	n/a	68	68	68
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	86
	ROM	n/a	n/a	n/a	n/a	n/a	76
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	90
	ROM	n/a	n/a	n/a	n/a	n/a	76
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	60	60	60	60	60	60
Category 2 ISR, floating-point	RAM	4	4	4	4	4	4
	ROM	86	86	86	86	86	86
Resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
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	24	24	24	24	24	24
Alarm	RAM	8	8	8	8	8	8
	ROM	40	40	40	40	40	40
Counter	RAM	2	2	2	2	2	2
	ROM	60	60	60	60	60	60
Message	RAM	11	11	11	31	31	31
	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	24	24	24	24	24	24

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	16	16	16	16	16	16
Arrivalpoint (writable)	RAM	16	16	16	16	16	16
	ROM	16	16	16	16	16	16
Schedule	RAM	12	12	12	12	12	12
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	8	8	8	8	8	8
Taskset (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8

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	8	8	8	8	8	8
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	12	12	12	12	12	12
	ROM	64	64	64	64	64	64
BCC2 task	RAM	n/a	16	18	n/a	16	18
	ROM	n/a	68	76	n/a	68	76
ECC1, Integer task	RAM	n/a	n/a	n/a	92	92	92
	ROM	n/a	n/a	n/a	84	84	84
ECC1, floating-point task	RAM	n/a	n/a	n/a	96	96	96
	ROM	n/a	n/a	n/a	84	84	84
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	94
	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	98
	ROM	n/a	n/a	n/a	n/a	n/a	92

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	8	8	8	8	8	8
	ROM	94	94	94	94	94	94
Category 2 ISR, floating-point	RAM	12	12	12	12	12	12
	ROM	116	116	116	116	116	116
Resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
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	24	24	24	24	24	24
Alarm	RAM	8	8	8	8	8	8
	ROM	40	40	40	40	40	40
Counter	RAM	2	2	2	2	2	2
	ROM	60	60	60	60	60	60
Message	RAM	11	11	11	31	31	31
	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	24	24	24	24	24	24
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	16	16	16	16	16	16
Arrivalpoint (writable)	RAM	16	16	16	16	16	16
	ROM	16	16	16	16	16	16
Schedule	RAM	12	12	12	12	12	12
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	8	8	8	8	8	8
Taskset (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	72	72	72	72	72	72
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	72	72	72	72	72	72
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	76	84	n/a	76	84
ECC1, Integer task	RAM	n/a	n/a	n/a	96	96	96
	ROM	n/a	n/a	n/a	92	92	92
ECC1, floating-point task	RAM	n/a	n/a	n/a	100	100	100
	ROM	n/a	n/a	n/a	92	92	92
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	98
	ROM	n/a	n/a	n/a	n/a	n/a	100
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	102
	ROM	n/a	n/a	n/a	n/a	n/a	100
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	106	106	106	106	106	106
Category 2 ISR, floating-point	RAM	16	16	16	16	16	16
	ROM	128	128	128	128	128	128
Resource	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
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	36	36	36	36	36	36
Alarm	RAM	8	8	8	8	8	8
	ROM	44	44	44	44	44	44
Counter	RAM	2	2	2	2	2	2
	ROM	64	64	64	64	64	64
Message	RAM	11	11	11	31	31	31
	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	36	36	36	36	36	36

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	28	28	28	28	28	28
Arrivalpoint (writable)	RAM	28	28	28	28	28	28
	ROM	28	28	28	28	28	28
Schedule	RAM	14	14	14	14	14	14
	ROM	42	42	42	42	42	42
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	8	8	8	8	8	8
Taskset (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8

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	164	274	324	202	312	422
	NS		128	236	286	172	276	386
	KL	2	100	208	272	142	246	370

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	22	22	22	22	22	22
ChainTask	SWL	1, 8	140	284	334	178	304	402
	SWH	1, 9	200	312	364	240	332	454
	NSL	8	140	284	334	178	304	402
	NSH	9	180	292	344	220	312	434
Schedule			160	160	218	160	160	218
GetTaskID			32	32	32	32	32	32
GetTaskState			116	116	116	146	146	146
EnableAllInterrupts			14	14	14	14	14	14
DisableAllInterrupts			22	22	22	22	22	22
ResumeAllInterrupts			30	30	30	30	30	30
SuspendAllInterrupts			48	48	48	48	48	48
ResumeOSInterrupts			30	30	30	30	30	30
SuspendOSInterrupts			42	42	42	42	42	42
GetResource	Task	7	100	100	140	100	100	140
	Combined	6	158	158	158	158	158	158
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	116	116	116	116	116	116
	Combined	6	202	202	202	202	202	202
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	208	208	368
	NS		n/a	n/a	n/a	170	170	332
	NS1i	10	n/a	n/a	n/a	64	n/a	n/a
	KL	2	n/a	n/a	n/a	142	142	302
	KL1i	2, 10	n/a	n/a	n/a	26	n/a	n/a
ClearEvent			n/a	n/a	n/a	48	48	48
GetEvent			n/a	n/a	n/a	16	16	16
WaitEvent	<default>		n/a	n/a	n/a	258	258	488
	fp	11	n/a	n/a	n/a	286	286	546
	1i	10	n/a	n/a	n/a	24	n/a	n/a
GetAlarmBase			85	85	85	85	85	85
GetAlarm			124	124	124	124	124	124
SetRelAlarm			172	172	172	172	172	172
SetAbsAlarm			192	192	192	192	192	192
CancelAlarm			92	92	92	92	92	92
InitCounter			78	78	78	78	78	78

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			86	86	86	86	86	86
osek_tick_alarm	<default>		102	102	102	102	102	102
	KL	2	44	44	44	44	44	44
osek_incr_counter			46	46	46	46	46	46
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			172	172	172	172	172	172
ShutdownOS	NoHook	12	14	14	14	14	14	14
	Hook	13	30	30	30	30	30	30
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			40	40	40	40	40	40
StopCOM			24	24	24	24	24	24
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	86	86	86	256	256	256
	CCCB	15	256	256	256	256	256	256
GetMessageResource			56	56	56	56	56	56
ReleaseMessageResource			54	54	54	54	54	54
GetMessageStatus			56	56	56	56	56	56
SendMessage	SW CCCA	1, 14	114	114	114	288	288	288
	SW CCCB	1, 15	272	272	272	288	288	288
	NS CCCA	14	114	114	114	288	288	288
	NS CCCB	15	272	272	272	288	288	288
	KL CCCA	2, 14	94	94	94	262	262	262
	KL CCCB	2, 15	246	246	246	262	262	262
main_dispatch	NoHook	12	348	348	444	348	348	444
	Hook	13	374	374	450	374	374	450
sub_dispatch	B1LF	19	50	50	50	50	50	50
	B1HI	20	124	124	124	124	124	124
	B1HF	21	132	132	132	132	132	132
	B2LI	22	n/a	112	144	n/a	112	144
	B2LF	23	n/a	120	152	n/a	120	152
	B2HI	24	n/a	196	264	n/a	196	264
	B2HF	25	n/a	204	272	n/a	204	272
	E1HI	26	n/a	n/a	n/a	414	414	510
	E1HF	27	n/a	n/a	n/a	422	422	518
	E2HI	28	n/a	n/a	n/a	n/a	n/a	510

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	518
ErrorHook support		16	58	58	58	58	58	58
	ServiceID	17	66	66	66	66	66	66
	Parameters	18	86	86	86	86	86	86
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	220	282	170	270	384
	NS		76	170	228	140	232	336
	KL	2	38	132	202	84	190	306
ChainTaskset	SWL	1, 8	112	182	242	112	206	278
	SWH	1, 9	126	266	336	126	270	376
	NSL	8	112	182	242	112	206	278
	NSH	9	106	246	316	106	250	356
GetTasksetRef			12	12	12	12	12	12
MergeTaskset			80	80	80	80	80	80
AssignTaskset			68	68	68	68	68	68
RemoveTaskset			106	106	106	106	106	106
TestSubTaskset			104	104	104	104	104	104
TestEquivalentTaskset			100	100	100	100	100	100
TickSchedule	SW	1	218	190	190	190	190	190
	NS		156	138	138	138	138	138
	KL	2	130	114	114	114	114	114
AdvanceSchedule	SW	1	194	182	182	182	182	182
	NS		142	130	130	130	130	130
	KL	2	116	104	104	104	104	104
StartSchedule			107	107	107	107	107	107
StopSchedule			78	78	78	78	78	78
GetScheduleStatus			118	118	118	118	118	118
GetScheduleValue			100	100	100	100	100	100
GetScheduleNext			16	16	16	16	16	16
SetScheduleNext			12	12	12	12	12	12
GetArrivalpointDelay			12	12	12	12	12	12
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			12	12	12	12	12	12
SetArrivalpointNext			8	8	8	8	8	8

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestArrivalpointWritable			32	32	32	32	32	32
GetExecutionTime			4	4	4	4	4	4
GetLargestExecutionTime			8	8	8	8	8	8
ResetLargestExecutionTime			4	4	4	4	4	4
GetStackOffset			32	32	32	32	32	32

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	164	274	324	202	312	422
	NS		128	236	286	172	276	386
	KL	2	100	208	272	142	246	370
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	22	22	22	22	22	22
ChainTask	SWL	1, 8	140	284	334	178	304	402
	SWH	1, 9	200	312	364	240	332	454
	NSL	8	140	284	334	178	304	402
	NSH	9	180	292	344	220	312	434
Schedule			182	182	240	182	182	240
GetTaskID			32	32	32	32	32	32
GetTaskState			116	116	116	146	146	146
EnableAllInterrupts			14	14	14	14	14	14
DisableAllInterrupts			22	22	22	22	22	22
ResumeAllInterrupts			30	30	30	30	30	30
SuspendAllInterrupts			48	48	48	48	48	48
ResumeOSInterrupts			30	30	30	30	30	30
SuspendOSInterrupts			42	42	42	42	42	42
GetResource	Task	7	100	100	140	100	100	140
	Combined	6	158	158	158	158	158	158
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	138	138	138	138	138	138
	Combined	6	246	246	246	246	246	246

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
SetEvent	SW	1	n/a	n/a	n/a	208	208	368
	NS		n/a	n/a	n/a	170	170	332
	NS1i	10	n/a	n/a	n/a	64	n/a	n/a
	KL	2	n/a	n/a	n/a	142	142	302
	KL1i	2, 10	n/a	n/a	n/a	26	n/a	n/a
ClearEvent			n/a	n/a	n/a	48	48	48
GetEvent			n/a	n/a	n/a	16	16	16
WaitEvent	<default>		n/a	n/a	n/a	322	322	542
	fp	11	n/a	n/a	n/a	350	350	598
	1i	10	n/a	n/a	n/a	124	n/a	n/a
GetAlarmBase			85	85	85	85	85	85
GetAlarm			124	124	124	124	124	124
SetRelAlarm			172	172	172	172	172	172
SetAbsAlarm			192	192	192	192	192	192
CancelAlarm			92	92	92	92	92	92
InitCounter			78	78	78	78	78	78
GetCounterValue			86	86	86	86	86	86
osek_tick_alarm	<default>		102	102	102	102	102	102
	KL	2	44	44	44	44	44	44
osek_incr_counter			46	46	46	46	46	46
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			236	236	236	236	236	236
ShutdownOS	NoHook	12	14	14	14	14	14	14
	Hook	13	30	30	30	30	30	30
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			40	40	40	40	40	40
StopCOM			24	24	24	24	24	24
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	86	86	86	256	256	256
	CCCB	15	256	256	256	256	256	256
GetMessageResource			56	56	56	56	56	56
ReleaseMessageResource			54	54	54	54	54	54
GetMessageStatus			56	56	56	56	56	56
SendMessage	SW CCCA	1, 14	114	114	114	288	288	288

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	SW CCCB	1, 15	272	272	272	288	288	288
	NS CCCA	14	114	114	114	288	288	288
	NS CCCB	15	272	272	272	288	288	288
	KL CCCA	2, 14	94	94	94	262	262	262
	KL CCCB	2, 15	246	246	246	262	262	262
main_dispatch	NoHook	12	408	408	492	408	408	492
	Hook	13	452	452	536	452	452	536
sub_dispatch	B1LF	19	26	26	26	26	26	26
	B1HI	20	122	122	122	122	122	122
	B1HF	21	130	130	130	130	130	130
	B2LI	22	n/a	84	116	n/a	84	116
	B2LF	23	n/a	92	124	n/a	92	124
	B2HI	24	n/a	192	260	n/a	192	260
	B2HF	25	n/a	200	268	n/a	200	268
	E1HI	26	n/a	n/a	n/a	448	448	544
	E1HF	27	n/a	n/a	n/a	456	456	552
	E2HI	28	n/a	n/a	n/a	n/a	n/a	544
	E2HF	29	n/a	n/a	n/a	n/a	n/a	552
ErrorHook support		16	58	58	58	58	58	58
	ServiceID	17	66	66	66	66	66	66
	Parameters	18	86	86	86	86	86	86
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	108	108	108	108	108	108
Timing_termination		4	76	76	76	76	76	76
ActivateTaskset	SW	1	126	220	282	170	270	384
	NS		76	170	228	140	232	336
	KL	2	38	132	202	84	190	306
ChainTaskset	SWL	1, 8	112	182	242	112	206	278
	SWH	1, 9	126	266	336	126	270	376
	NSL	8	112	182	242	112	206	278
	NSH	9	106	246	316	106	250	356
GetTasksetRef			12	12	12	12	12	12
MergeTaskset			80	80	80	80	80	80
AssignTaskset			68	68	68	68	68	68
RemoveTaskset			106	106	106	106	106	106
TestSubTaskset			104	104	104	104	104	104
TestEquivalentTaskset			100	100	100	100	100	100

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TickSchedule	SW	1	218	190	190	190	190	190
	NS		156	138	138	138	138	138
	KL	2	130	114	114	114	114	114
AdvanceSchedule	SW	1	194	182	182	182	182	182
	NS		142	130	130	130	130	130
	KL	2	116	104	104	104	104	104
StartSchedule			107	107	107	107	107	107
StopSchedule			78	78	78	78	78	78
GetScheduleStatus			118	118	118	118	118	118
GetScheduleValue			100	100	100	100	100	100
GetScheduleNext			16	16	16	16	16	16
SetScheduleNext			12	12	12	12	12	12
GetArrivalpointDelay			12	12	12	12	12	12
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			12	12	12	12	12	12
SetArrivalpointNext			8	8	8	8	8	8
TestArrivalpointWritable			32	32	32	32	32	32
GetExecutionTime			116	116	116	116	116	116
GetLargestExecutionTime			20	20	20	20	20	20
ResetLargestExecutionTime			16	16	16	16	16	16
GetStackOffset			32	32	32	32	32	32

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	264	372	424	292	408	506
	NS		358	476	524	398	512	598
	KL	2	220	334	382	260	370	468
TerminateTask	LExt	3	126	126	126	126	126	126
	H	5	180	180	180	180	180	180
ChainTask	SWL	1, 8	286	404	452	306	422	536

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	SWH	1, 9	368	468	518	392	502	624
	NSL	8	402	540	586	444	560	654
	NSH	9	462	584	630	508	604	710
Schedule			274	274	332	274	274	332
GetTaskID			52	52	52	52	52	52
GetTaskState			246	246	246	262	262	262
EnableAllInterrupts			34	34	34	34	34	34
DisableAllInterrupts			46	46	46	46	46	46
ResumeAllInterrupts			98	98	98	98	98	98
SuspendAllInterrupts			72	72	72	72	72	72
ResumeOSInterrupts			98	98	98	98	98	98
SuspendOSInterrupts			66	66	66	66	66	66
GetResource	Task	7	390	390	382	390	390	382
	Combined	6	394	394	394	394	394	394
	CLEx	3	346	346	346	346	346	346
ReleaseResource	Task	7	358	358	358	358	358	358
	Combined	6	480	480	480	480	480	480
	CLEx	3	310	310	310	310	310	310
SetEvent	SW	1	n/a	n/a	n/a	366	366	514
	NS		n/a	n/a	n/a	438	438	594
	NS1i	10	n/a	n/a	n/a	214	n/a	n/a
	KL	2	n/a	n/a	n/a	304	304	438
	KL1i	2, 10	n/a	n/a	n/a	198	n/a	n/a
ClearEvent			n/a	n/a	n/a	162	162	162
GetEvent			n/a	n/a	n/a	232	232	232
WaitEvent	<default>		n/a	n/a	n/a	450	450	654
	fp	11	n/a	n/a	n/a	476	476	708
	1i	10	n/a	n/a	n/a	254	n/a	n/a
GetAlarmBase			173	173	173	173	173	173
GetAlarm			174	174	174	174	174	174
SetRelAlarm			244	244	244	244	244	244
SetAbsAlarm			270	270	270	270	270	270
CancelAlarm			146	146	146	146	146	146
InitCounter			216	216	216	216	216	216
GetCounterValue			180	180	180	180	180	180
osek_tick_alarm	<default>		102	102	102	102	102	102
	KL	2	44	44	44	44	44	44

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
osek_incr_counter			46	46	46	46	46	46
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			260	260	260	260	260	260
ShutdownOS	NoHook	12	26	26	26	26	26	26
	Hook	13	42	42	42	42	42	42
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			64	64	64	64	64	64
StopCOM			58	58	58	58	58	58
ReadFlag			38	38	38	38	38	38
ResetFlag			40	40	40	40	40	40
ReceiveMessage	CCCA	14	174	174	174	330	330	330
	CCCB	15	330	330	330	330	330	330
GetMessageResource			122	122	122	122	122	122
ReleaseMessageResource			130	130	130	130	130	130
GetMessageStatus			132	132	132	132	132	132
SendMessage	SW CCCA	1, 14	196	196	196	356	356	356
	SW CCCB	1, 15	340	340	340	356	356	356
	NS CCCA	14	196	196	196	356	356	356
	NS CCCB	15	340	340	340	356	356	356
	KL CCCA	2, 14	190	190	190	348	348	348
	KL CCCB	2, 15	330	330	330	348	348	348
main_dispatch	NoHook	12	408	408	492	408	408	492
	Hook	13	452	452	536	452	452	536
sub_dispatch	B1LF	19	26	26	26	26	26	26
	B1HI	20	122	122	122	122	122	122
	B1HF	21	130	130	130	130	130	130
	B2LI	22	n/a	84	116	n/a	84	116
	B2LF	23	n/a	92	124	n/a	92	124
	B2HI	24	n/a	192	260	n/a	192	260
	B2HF	25	n/a	200	268	n/a	200	268
	E1HI	26	n/a	n/a	n/a	448	448	544
	E1HF	27	n/a	n/a	n/a	456	456	552
	E2HI	28	n/a	n/a	n/a	n/a	n/a	544
	E2HF	29	n/a	n/a	n/a	n/a	n/a	552
ErrorHook support		16	104	104	104	104	104	104
	ServiceID	17	124	124	124	124	124	124

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Parameters	18	176	176	176	176	176	176
validity_checks		3	23	23	23	23	23	23
Timing_dispatch		4	108	108	108	108	108	108
Timing_termination		4	76	76	76	76	76	76
ActivateTaskset	SW	1	350	482	540	422	540	634
	NS		434	560	624	514	648	736
	KL	2	286	418	494	368	506	602
ChainTaskset	SWL	1, 8	460	596	656	480	628	714
	SWH	1, 9	542	678	736	568	720	814
	NSL	8	608	716	776	628	766	872
	NSH	9	650	770	828	676	828	924
GetTasksetRef			172	172	172	172	172	172
MergeTaskset			280	280	280	280	280	280
AssignTaskset			254	254	254	254	254	254
RemoveTaskset			294	294	294	294	294	294
TestSubTaskset			296	296	296	296	296	296
TestEquivalentTaskset			292	292	292	292	292	292
TickSchedule	SW	1	430	296	296	296	296	296
	NS		524	400	400	400	400	400
	KL	2	374	232	232	232	232	232
AdvanceSchedule	SW	1	432	300	300	300	300	300
	NS		502	400	400	400	400	400
	KL	2	392	248	248	248	248	248
StartSchedule			247	247	247	247	247	247
StopSchedule			176	176	176	176	176	176
GetScheduleStatus			212	212	212	212	212	212
GetScheduleValue			196	196	196	196	196	196
GetScheduleNext			112	112	112	112	112	112
SetScheduleNext			212	212	212	212	212	212
GetArrivalpointDelay			162	162	162	162	162	162
SetArrivalpointDelay			182	182	182	182	182	182
GetArrivalpointTasksetRef			158	158	158	158	158	158
GetArrivalpointNext			162	162	162	162	162	162
SetArrivalpointNext			226	226	226	226	226	226
TestArrivalpointWritable			174	174	174	174	174	174
GetExecutionTime			138	138	138	138	138	138
GetLargestExecutionTime			148	148	148	148	148	148

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
ResetLargestExecutionTime			132	132	132	132	132	132
GetStackOffset			32	32	32	32	32	32

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` 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

Number	Note
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	44	93	115	46	89	135
	NS	31	87	110	37	84	131
	KL	29	85	105	35	82	131
TerminateTask	LExt	0	0	0	0	0	0
	H	76	72	80	76	76	81
ChainTask	SWL	144	199	260	184	232	308
	SWH	179	220	284	215	250	334
	NSL	144	200	263	178	227	307
	NSH	177	218	281	213	247	330
Schedule	SW	51	55	66	51	51	62
GetTaskID		20	20	18	20	18	17
GetTaskState		36	36	36	46	46	46

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
EnableAllInterrupts		7	7	8	6	6	6
DisableAllInterrupts		12	8	12	7	7	11
ResumeAllInterrupts		10	10	11	9	9	9
SuspendAllInterrupts		13	13	13	12	12	12
ResumeOSInterrupts		10	10	11	9	9	9
SuspendOSInterrupts		13	13	13	12	12	12
GetResource	Task	37	37	44	34	34	45
	Combined	29	29	29	30	30	30
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	29	29	29	30	30	30
	Combined	44	44	43	45	45	45
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	51	47	54
	NS	n/a	n/a	n/a	46	46	49
	KL	n/a	n/a	n/a	49	49	51
ClearEvent		n/a	n/a	n/a	22	22	22
GetEvent		n/a	n/a	n/a	15	15	15
WaitEvent	<default>	n/a	n/a	n/a	213	213	263
	fp	n/a	n/a	n/a	219	219	270
GetAlarmBase		35	31	35	35	35	35
GetAlarm		34	39	39	34	38	38
SetRelAlarm		41	42	42	41	45	41
SetAbsAlarm		40	39	39	43	39	39
CancelAlarm		27	22	22	26	22	22
InitCounter		25	29	29	25	29	30
GetCounterValue		29	25	25	29	25	25
osek_tick_alarm	<default>	29	29	29	33	29	29
	KL	22	25	21	22	26	26
osek_incr_counter		11	11	11	10	6	10
GetActiveApplicationMode		6	6	6	7	3	7
StartOS		913	961	913	965	961	965
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	11	11	15	10	10	14
InitCOM		6	10	10	9	9	9
CloseCOM		10	6	6	9	5	5
StartCOM		23	24	19	70	74	74

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
StopCOM		15	11	11	15	11	11
ReadFlag		n/a	n/a	n/a	11	7	7
ResetFlag		n/a	n/a	n/a	10	6	6
ReceiveMessage		27	24	23	140	140	140
GetMessageResource		n/a	n/a	n/a	70	66	66
ReleaseMessageResource		n/a	n/a	n/a	57	61	61
GetMessageStatus		n/a	n/a	n/a	29	29	29
SendMessage	SW	80	131	153	193	241	287
	NS	66	123	146	190	228	275
	KL	66	112	140	180	227	276
ActivateTaskset	SW	30	653	751	32	694	774
	NS	16	679	678	25	686	769
	KL	15	638	738	18	681	734
	SW2	26	653	751	32	694	778
	NS2	16	679	678	25	686	769
	KL2	11	642	734	22	677	730
ChainTaskset	SWL	139	735	799	173	798	851
	SWH	173	766	871	205	802	984
	NSL	143	731	835	173	798	882
	NSH	167	767	832	198	795	945
GetTasksetRef		11	11	15	15	15	11
MergeTaskset		24	24	24	19	19	19
AssignTaskset		20	20	20	15	15	15
RemoveTaskset		26	25	26	20	20	20
TestSubTaskset		27	27	27	22	22	22
TestEquivalentTaskset		25	25	25	21	21	21
TickSchedule	SW	64	698	794	78	749	793
	NS	49	688	784	72	739	787
	KL	50	689	781	69	736	784
	SW2	64	702	798	82	741	793
	NS2	49	692	788	68	731	784
	KL2	50	685	785	65	728	781
AdvanceSchedule	SW	54	693	785	73	740	788
	NS	45	684	775	59	726	774
	KL	36	676	772	57	724	772
	SW2	54	693	785	73	732	785

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NS2	41	684	775	59	722	775
	KL2	40	676	772	57	720	772
StartSchedule		42	42	42	43	43	43
StopSchedule		31	31	32	31	31	31
GetScheduleStatus		38	38	38	37	37	37
GetScheduleValue		31	31	32	32	32	32
GetScheduleNext		11	11	12	16	16	16
SetScheduleNext		11	11	11	15	15	15
GetArrivalpointDelay		15	16	15	11	11	11
SetArrivalpointDelay		11	12	11	8	8	8
GetArrivalpointTasksetRef		7	7	7	12	12	12
GetArrivalpointNext		7	8	7	13	13	13
SetArrivalpointNext		7	8	7	12	12	12
TestArrivalpointWritable		17	17	17	12	12	12
GetExecutionTime		10	6	10	5	5	9
GetLargestExecutionTime		10	10	10	9	9	9
ResetLargestExecutionTime		9	9	9	8	8	8
GetStackOffset		11	11	11	11	11	11

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	44	93	116	46	89	140
	NS	32	87	109	41	84	127
	KL	30	85	105	35	78	126
TerminateTask	LExt	0	0	0	0	0	0
	H	175	163	174	173	173	173
ChainTask	SWL	259	321	383	302	356	431
	SWH	294	326	385	330	365	453
	NSL	259	324	378	298	347	430
	NSH	280	332	391	327	361	441

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Schedule	SW	55	51	62	51	51	66
GetTaskID		21	17	21	16	20	18
GetTaskState		36	36	36	46	46	42
EnableAllInterrupts		7	7	8	7	7	6
DisableAllInterrupts		8	12	8	7	7	7
ResumeAllInterrupts		10	10	11	10	10	9
SuspendAllInterrupts		13	13	13	12	12	12
ResumeOSInterrupts		10	10	11	10	10	9
SuspendOSInterrupts		13	13	13	12	12	12
GetResource	Task	37	37	44	38	38	45
	Combined	29	29	29	30	30	30
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	29	29	29	30	30	30
	Combined	44	44	44	45	45	45
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	47	47	49
	NS	n/a	n/a	n/a	46	46	48
	KL	n/a	n/a	n/a	45	45	51
ClearEvent		n/a	n/a	n/a	18	18	18
GetEvent		n/a	n/a	n/a	15	15	15
WaitEvent	<default>	n/a	n/a	n/a	310	310	359
	fp	n/a	n/a	n/a	321	321	369
GetAlarmBase		35	31	31	31	31	35
GetAlarm		39	34	34	38	38	35
SetRelAlarm		42	41	41	41	41	42
SetAbsAlarm		39	44	44	43	43	38
CancelAlarm		22	23	23	26	26	25
InitCounter		30	26	26	29	29	25
GetCounterValue		25	29	29	25	25	29
osek_tick_alarm	<default>	33	33	33	33	33	33
	KL	25	22	26	22	22	21
osek_incr_counter		7	7	7	6	6	6
GetActiveApplicationMode		2	6	6	7	7	3
StartOS		1651	1744	1655	1651	1655	1744
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	15	15	11	14	14	10

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
InitCOM		6	10	10	9	9	5
CloseCOM		6	10	10	9	9	5
StartCOM		24	23	20	70	70	69
StopCOM		15	11	11	11	11	15
ReadFlag		n/a	n/a	n/a	11	11	7
ResetFlag		n/a	n/a	n/a	6	6	10
ReceiveMessage		24	27	28	140	144	135
GetMessageResource		n/a	n/a	n/a	70	70	66
ReleaseMessageResource		n/a	n/a	n/a	60	60	61
GetMessageStatus		n/a	n/a	n/a	29	33	29
SendMessage	SW	78	128	151	193	237	295
	NS	68	122	144	190	232	283
	KL	65	114	142	184	227	278
ActivateTaskset	SW	30	684	715	36	694	742
	NS	20	706	705	29	686	733
	KL	11	674	702	18	677	760
	SW2	26	688	719	32	690	746
	NS2	20	706	705	29	686	733
	KL2	15	670	706	22	681	764
ChainTaskset	SWL	258	883	951	295	920	1004
	SWH	284	908	945	316	916	1066
	NSL	258	883	912	295	920	975
	NSH	281	907	969	318	910	1095
GetTasksetRef		11	11	15	15	15	11
MergeTaskset		24	24	24	19	19	19
AssignTaskset		20	20	20	15	15	15
RemoveTaskset		25	26	25	20	20	21
TestSubTaskset		27	27	27	22	22	22
TestEquivalentTaskset		25	25	25	21	21	21
TickSchedule	SW	63	730	762	82	749	824
	NS	53	720	752	72	739	814
	KL	45	721	749	65	732	811
	SW2	63	730	762	82	741	820
	NS2	53	720	752	72	731	810
	KL2	45	721	749	65	724	807
AdvanceSchedule	SW	53	720	752	69	736	820

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NS	41	716	748	63	730	805
	KL	40	707	744	57	724	803
	SW2	53	720	752	69	728	816
	NS2	41	716	748	63	722	801
	KL2	40	707	744	57	716	799
StartSchedule		42	42	42	43	43	43
StopSchedule		32	32	31	31	31	32
GetScheduleStatus		38	38	38	37	37	37
GetScheduleValue		32	32	31	32	32	33
GetScheduleNext		12	12	11	16	16	17
SetScheduleNext		11	11	11	15	15	15
GetArrivalpointDelay		16	15	16	11	11	10
SetArrivalpointDelay		12	11	12	8	8	7
GetArrivalpointTasksetRef		7	7	7	12	12	12
GetArrivalpointNext		8	7	8	13	13	12
SetArrivalpointNext		8	7	8	12	12	11
TestArrivalpointWritable		17	17	17	12	12	12
GetExecutionTime		51	54	51	45	45	49
GetLargestExecutionTime		18	19	18	18	18	17
ResetLargestExecutionTime		17	17	17	16	16	16
GetStackOffset		11	11	11	11	11	11

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	175	219	238	178	218	248
	NS	204	258	277	209	249	285
	KL	163	216	234	172	212	242
TerminateTask	LExt	181	191	200	185	192	197
	H	212	214	214	203	210	223
ChainTask	SWL	409	481	530	452	493	577

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	SWH	447	481	532	481	519	577
	NSL	453	514	569	494	538	616
	NSH	477	525	579	514	550	621
Schedule	SW	72	68	80	72	72	80
GetTaskID		23	23	21	23	19	21
GetTaskState		176	172	176	171	170	171
EnableAllInterrupts		9	9	9	10	10	10
DisableAllInterrupts		10	10	14	15	15	11
ResumeAllInterrupts		18	18	18	19	19	19
SuspendAllInterrupts		15	15	15	16	15	16
ResumeOSInterrupts		18	18	18	19	19	19
SuspendOSInterrupts		15	15	15	16	15	16
GetResource	Task	329	329	189	357	367	221
	Combined	145	145	145	178	176	178
	CLEx	181	185	185	212	212	216
ReleaseResource	Task	159	159	159	192	192	192
	Combined	142	142	142	175	175	175
	CLEx	162	162	162	192	192	192
SetEvent	SW	n/a	n/a	n/a	193	192	189
	NS	n/a	n/a	n/a	198	197	200
	KL	n/a	n/a	n/a	187	186	187
ClearEvent		n/a	n/a	n/a	39	44	40
GetEvent		n/a	n/a	n/a	174	173	174
WaitEvent	<default>	n/a	n/a	n/a	363	367	397
	fp	n/a	n/a	n/a	370	372	407
GetAlarmBase		119	115	115	119	118	114
GetAlarm		121	117	117	117	116	121
SetRelAlarm		137	141	141	141	140	143
SetAbsAlarm		131	135	135	130	129	136
CancelAlarm		110	106	106	105	104	104
InitCounter		208	204	207	204	206	211
GetCounterValue		99	103	102	103	101	98
osek_tick_alarm	<default>	29	33	33	29	29	29
	KL	22	26	26	22	22	25
osek_incr_counter		7	11	11	10	10	10
GetActiveApplicationMode		2	6	6	3	3	7

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
StartOS		1786	1786	1790	1786	1895	1790
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	16	16	12	16	16	16
InitCOM		6	10	10	5	5	9
CloseCOM		6	10	10	5	5	9
StartCOM		28	24	24	75	76	79
StopCOM		20	16	16	19	20	15
ReadFlag		n/a	n/a	n/a	13	13	17
ResetFlag		n/a	n/a	n/a	13	13	17
ReceiveMessage		85	81	85	196	199	196
GetMessageResource		n/a	n/a	n/a	290	286	293
ReleaseMessageResource		n/a	n/a	n/a	262	265	263
GetMessageStatus		n/a	n/a	n/a	88	88	89
SendMessage	SW	267	310	329	380	419	458
	NS	297	350	369	413	452	496
	KL	260	305	323	370	409	456
ActivateTaskset	SW	618	1285	1315	687	1380	1401
	NS	641	1339	1370	713	1438	1397
	KL	603	1317	1285	713	1416	1343
	SW2	614	1289	1319	691	1384	1397
	NS2	641	1339	1370	713	1438	1397
	KL2	607	1313	1281	709	1412	1347
ChainTaskset	SWL	922	1532	1599	991	1622	1685
	SWH	980	1621	1587	1075	1748	1775
	NSL	992	1667	1661	962	1624	1828
	NSH	975	1744	1775	984	1718	1840
GetTasksetRef		146	146	150	150	149	150
MergeTaskset		63	63	63	58	58	58
AssignTaskset		59	57	59	52	54	54
RemoveTaskset		63	64	63	59	58	58
TestSubTaskset		63	63	63	58	58	58
TestEquivalentTaskset		60	60	60	56	56	56
TickSchedule	SW	122	1394	1362	789	1509	1437
	NS	159	1429	1397	825	1544	1473
	KL	109	1389	1357	784	1503	1424
	SW2	122	1394	1362	789	1494	1427

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NS2	159	1429	1397	825	1529	1462
	KL2	109	1389	1357	784	1488	1414
AdvanceSchedule	SW	113	1387	1355	787	1506	1427
	NS	142	1428	1396	818	1537	1465
	KL	108	1381	1349	782	1500	1421
	SW2	113	1387	1355	787	1491	1416
	NS2	142	1428	1396	818	1522	1454
	KL2	108	1381	1349	782	1485	1411
StartSchedule		73	73	73	74	74	74
StopSchedule		45	45	45	45	43	45
GetScheduleStatus		51	51	51	50	45	50
GetScheduleValue		45	45	45	46	45	46
GetScheduleNext		25	25	25	30	31	30
SetScheduleNext		55	55	55	59	59	59
GetArrivalpointDelay		41	41	41	36	36	36
SetArrivalpointDelay		50	50	50	46	45	46
GetArrivalpointTasksetRef		27	27	27	32	32	32
GetArrivalpointNext		28	28	28	33	33	33
SetArrivalpointNext		54	54	54	58	58	58
TestArrivalpointWritable		36	36	36	31	30	31
GetExecutionTime		53	49	49	52	56	56
GetLargestExecutionTime		138	138	138	136	136	136
ResetLargestExecutionTime		135	135	135	133	133	133
GetStackOffset		11	11	11	11	11	11

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	31	31	31	31	31	31
	Cat 2	40	40	40	39	39	39

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	31	31	31	31	31	31
	Cat 2	116	120	120	119	119	115

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	31	31	31	31	31	31
	Cat 2	122	125	125	120	120	123

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.

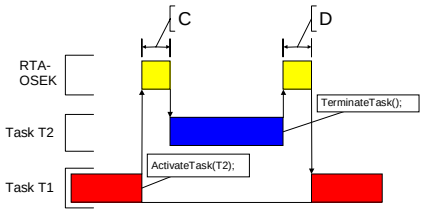

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

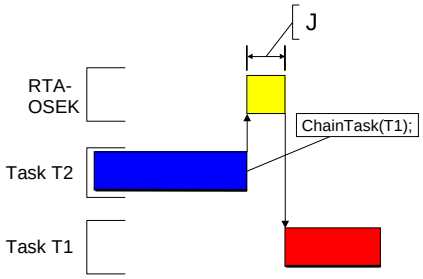

Figure 2: Task Chaining

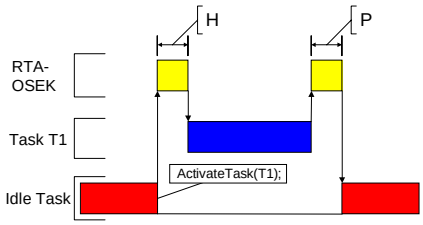

Figure 3: Task Activation from Idle Task

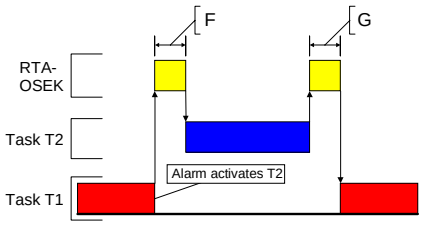

Figure 4: Task Activation from an Alarm

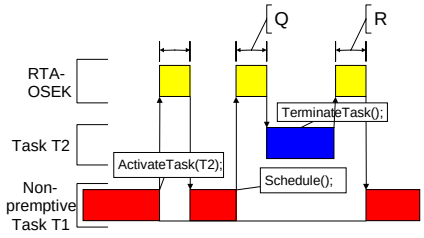

Figure 5: Non-Premptive Task Calls Schedule()

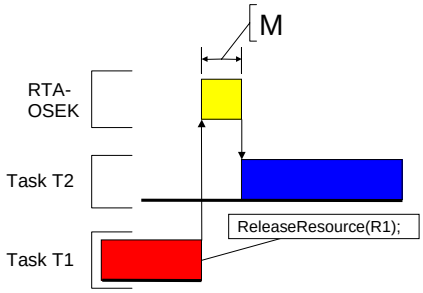

Figure 6: Blocked Task Activated by ReleaseResource()

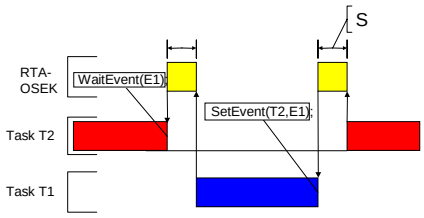

Figure 7: Waiting Task Activated by SetEvent()

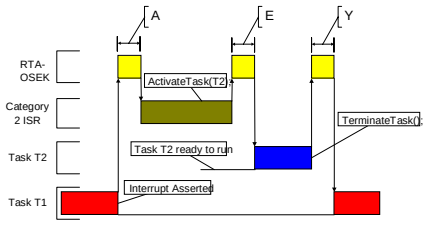

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	36	52	70	38	52	72
Figure 1: D	Heavy, Basic/Extended	73	83	101	98	98	117
ChainTask	Light, Basic	117	186	243	119	188	268
Figure 2: J	Heavy, Basic/Extended	241	315	384	273	320	422
Pre-emption	Light, Basic	94	156	220	98	158	246
Figure 1: C	Heavy, Basic/Extended	129	182	245	167	210	298
From idle task	Light, Basic	94	153	220	94	158	246
Figure 3: H	Heavy, Basic/Extended	126	182	246	167	206	294
Triggered by alarm	Light, Basic	131	193	257	135	194	282
Figure 4: F	Heavy, Basic/Extended	164	216	280	201	245	332
Schedule	Light, Basic	86	101	149	87	101	153
Figure 5: Q	Heavy, Basic/Extended	126	122	178	155	155	208
Release resource	Light, Basic	91	97	142	87	101	139
Figure 6: M	Heavy, Basic/Extended	126	123	168	156	160	201
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	275	275	385
From category 2 ISR	Light, Basic	78	94	135	78	92	134
Figure 8: E	Heavy, Basic/Extended	114	119	160	147	151	192

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	142	147	169	139	157	170
Figure 1: D	Heavy, Basic/Extended	172	178	190	197	197	215
ChainTask	Light, Basic	235	310	361	243	308	382
Figure 2: J	Heavy, Basic/Extended	456	509	590	488	526	628
Pre-emption	Light, Basic	183	235	300	184	241	321
Figure 1: C	Heavy, Basic/Extended	211	261	326	243	291	376

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
From idle task	Light, Basic	179	240	296	180	241	317
Figure 3: H	Heavy, Basic/Extended	203	262	318	239	291	376
Triggered by alarm	Light, Basic	220	272	337	225	273	362
Figure 4: F	Heavy, Basic/Extended	241	300	356	282	321	406
Schedule	Light, Basic	176	175	224	177	184	228
Figure 5: Q	Heavy, Basic/Extended	204	201	250	236	240	289
Release resource	Light, Basic	176	180	222	177	176	217
Figure 6: M	Heavy, Basic/Extended	204	206	248	236	232	270
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	340	340	451
From category 2 ISR	Light, Basic	264	271	309	259	266	319
Figure 8: E	Heavy, Basic/Extended	292	295	329	318	322	374

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	179	203	217	178	203	218
Figure 1: D	Heavy, Basic/Extended	209	220	243	239	238	262
ChainTask	Light, Basic	398	454	514	401	452	530
Figure 2: J	Heavy, Basic/Extended	654	701	789	692	717	805
Pre-emption	Light, Basic	305	356	418	308	356	433
Figure 1: C	Heavy, Basic/Extended	329	378	440	372	406	481
From idle task	Light, Basic	306	357	414	308	352	429
Figure 3: H	Heavy, Basic/Extended	334	383	432	368	406	477
Triggered by alarm	Light, Basic	342	393	455	349	397	465
Figure 4: F	Heavy, Basic/Extended	368	417	470	411	445	511
Schedule	Light, Basic	190	194	248	195	194	240
Figure 5: Q	Heavy, Basic/Extended	214	216	270	259	250	295
Release resource	Light, Basic	288	296	338	324	327	370
Figure 6: M	Heavy, Basic/Extended	312	318	360	388	383	425
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	495	500	590

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	268	274	312	275	279	306
Figure 8: E	Heavy, Basic/Extended	290	294	332	333	333	361

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		128	128	136	128	128	136
BCC1 lightweight, floating-point		136	136	144	136	136	144
BCC1 heavyweight, integer		208	208	216	208	208	216
BCC1 heavyweight, floating-point		208	208	216	208	208	216
BCC2 lightweight, integer		n/a	136	144	n/a	136	144
BCC2 lightweight, floating-point		n/a	136	144	n/a	136	144
BCC2 heavyweight, integer		n/a	208	216	n/a	208	216
BCC2 heavyweight, floating-point		n/a	208	216	n/a	208	216
ECC1 heavyweight, integer		n/a	n/a	n/a	232	232	240
ECC1 heavyweight, floating-point		n/a	n/a	n/a	232	232	240
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	240
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	240
Pre- and/or Post-Task hooks used							

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Task type						
BCC1 lightweight, integer	120	120	120	120	120	120
BCC1 lightweight, floating-point	128	128	128	128	128	128
BCC1 heavyweight, integer	200	200	200	200	200	200
BCC1 heavyweight, floating-point	200	200	200	200	200	200
BCC2 lightweight, integer	n/a	128	128	n/a	128	128
BCC2 lightweight, floating-point	n/a	128	128	n/a	128	128
BCC2 heavyweight, integer	n/a	200	200	n/a	200	200
BCC2 heavyweight, floating-point	n/a	200	200	n/a	200	200
ECC1 heavyweight, integer	n/a	n/a	n/a	224	224	224
ECC1 heavyweight, floating-point	n/a	n/a	n/a	224	224	224
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	224
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	224

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	136	136	140	136	136	140
BCC1 lightweight, floating-point	140	140	144	140	140	144
BCC1 heavyweight, integer	212	212	216	212	212	216
BCC1 heavyweight, floating-point	212	212	216	212	212	216
BCC2 lightweight, integer	n/a	140	144	n/a	140	144
BCC2 lightweight, floating-point	n/a	140	144	n/a	140	144
BCC2 heavyweight, integer	n/a	216	220	n/a	216	220
BCC2 heavyweight, floating-point	n/a	216	220	n/a	216	220
ECC1 heavyweight, integer	n/a	n/a	n/a	236	236	240
ECC1 heavyweight, floating-point	n/a	n/a	n/a	236	236	240
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	240
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	240

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	136	136	140	136	136	140
BCC1 lightweight, floating-point	140	140	144	140	140	144
BCC1 heavyweight, integer	212	212	216	212	212	216
BCC1 heavyweight, floating-point	212	212	216	212	212	216
BCC2 lightweight, integer	n/a	140	144	n/a	140	144
BCC2 lightweight, floating-point	n/a	140	144	n/a	140	144
BCC2 heavyweight, integer	n/a	216	220	n/a	216	220
BCC2 heavyweight, floating-point	n/a	216	220	n/a	216	220
ECC1 heavyweight, integer	n/a	n/a	n/a	236	236	240
ECC1 heavyweight, floating-point	n/a	n/a	n/a	236	236	240
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	240
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	240

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	136	136	140	136	136	140
BCC1 lightweight, floating-point	140	140	144	140	140	144
BCC1 heavyweight, integer	212	212	216	212	212	216
BCC1 heavyweight, floating-point	212	212	216	212	212	216
BCC2 lightweight, integer	n/a	140	144	n/a	140	144
BCC2 lightweight, floating-point	n/a	140	144	n/a	140	144
BCC2 heavyweight, integer	n/a	216	220	n/a	216	220
BCC2 heavyweight, floating-point	n/a	216	220	n/a	216	220
ECC1 heavyweight, integer	n/a	n/a	n/a	244	244	248
ECC1 heavyweight, floating-point	n/a	n/a	n/a	244	244	248
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	248
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	248

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		136	136	140	136	136	140
BCC1 lightweight, floating-point		140	140	144	140	140	144
BCC1 heavyweight, integer		212	212	216	212	212	216
BCC1 heavyweight, floating-point		212	212	216	212	212	216
BCC2 lightweight, integer		n/a	140	144	n/a	140	144
BCC2 lightweight, floating-point		n/a	140	144	n/a	140	144
BCC2 heavyweight, integer		n/a	216	220	n/a	216	220
BCC2 heavyweight, floating-point		n/a	216	220	n/a	216	220
ECC1 heavyweight, integer		n/a	n/a	n/a	244	244	248
ECC1 heavyweight, floating-point		n/a	n/a	n/a	244	244	248
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	248
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	248

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.