RTA-OSEK

Binding Manual: SH2A/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: M00081-001

Disclaimer

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

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

Trademarks

RTA-OSEK 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 SH2A/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	ccsh.exe
Version	4.0.8D

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

Option	Description
`-cpu=sh2a`	Compile for the SH-2A/SH2A-FPU

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

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

Option	Description
`-cpu=sh2a`	Compile for the SH-2A/SH2A-FPU
`--no_debug`	Turn debug mode off
`-Onone`	Turn the optimizer off

2.2 Assembler

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

Vendor	Green Hills Software Inc.
Assembler	assh.exe
Version	4.0.8D

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

Option	Description
`-cpu=sh2a`	Assemble for the SH-2A/SH2A-CPU

The assembly file that RTA-OSEK generates from your OIL configuration file is called `osgen.sh`. 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
`os_vectbl`	ROM	Vector table if generated by RTA-OSEK GUI
`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_text`	ROM	RTA-OSEK code
`os_trace_ram`	RAM	RTA-TRACE buffer

In some cases, sections produced by the linker must be located according to special constraints. The following table indicates which sections must be located with which particular constraints:

Sections	Constraints
`.stack`	For ECC support, stack must be placed in the section named `.stack`

2.3.1 Linking SH-2A/SH2A-FPU applications with the RTA-OSEK libraries

The RTA-OSEK runtime libraries are compiled to create object code to run on SH-2A processor cores. As the RTA-OSEK runtime libraries do not use the floating-point hardware, the resultant libraries can be used with either SH-2A or SH2A-FPU processors. Applications have been successfully tested on both CPU derivatives.

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 *SH-2A, SH2-FPU Software Manual*.

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

IPL value	Status Register (SR), bits 4-7	Description
0	0	User level
1-15	0001 - 1111	Category 1 and 2 interrupts
16	Non-maskable interrupts, NMI, traps etc	Category 1 interrupts only

3.1.2 Interrupt Vectors

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

Vector	Legality
0-3	Reset vectors; must be defined by the user outside RTA-OSEK
7, 8, 19-31	Reserved

The valid base addresses for the vector table are:

Base Address	Notes
VBR	Aligned to 4-byte boundary. Vector Base Register must be set before `StartOS()` is called.

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

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

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
4-511	_os_wrapper_VVVV
eg :	_os_wrapper_02F0

The SH-2A/SH2A-FPU has a vector table that may be placed at any location in memory with an address that is aligned to a 4-byte boundary. The vector table can be provided by RTA-OSEK or user defined. RTA-OSEK permits vectors to be placed in the range 4 to 511 (offsets 0x010 to 0x7FC).

The reset values for the processor registers defined in vectors 0-3 (offsets 0x0 to 0xC) must be provided and linked by the user (demonstrated in the RTA-OSEK example application).

3.1.6 Interrupt Priority Registers

The SH-2A/SH2A-FPU interrupt controller contains a set of interrupt priority registers (IPRs) that are used to specify the priority level that processor interrupts operate.

To permit user control over enabling and disabling of hardware interrupts, the RTA-OSEK Component does not initialize the peripheral interrupt enable registers (e.g. `TIER1A`, `TIER1B`, etc.), or the IPRs.

Important: The correct initialization of an interrupt source and the associated interrupt priority level in the IPRs is left as the responsibility of the user. Each IPR must be programmed with the interrupt priority level specified in the RTA-OSEK GUI.

As the interrupt priorities for Category 1 and 2 interrupts are specified in the RTA-OSEK GUI, RTA-OSEK generates the values for the IPR registers. The values are made available to user code, to be copied to the registers during system initialization, as an array of 16-bit values called `os_ipr_values` in the file `osekdefs.c`. The array is conditionally compiled; if it is required then the following compiler command line option should be used (as demonstrated in the example application):

```
-DOS_IPL_INIT
```

Portability: The number of IPRs varies according to the SH-2A/SH2A-FPU processor. The RTA-OSEK GUI always generates the IPRs correct for the target variant selected. If the Generic SH2A variant is used then care must be taken when programming IPRs that the values are correct for the processor used – refer to the Interrupt Controller Section of the *Renesas Hardware Manual* for the relevant SH-2A/SH2A-FPU processor for a list of valid IPRs and the vector numbers they represent.

Care must be taken when setting the priorities of vectors that share the same bits in an IPR. The RTA-OSEK GUI checks that vector priorities are consistent during the "Build checks" which precede building an application.

3.1.7 Category 1 ISRs

Category 1 interrupts can be declared at priorities 1 to 15 in the RTA-OSEK GUI with the exception of the following priorities:

Permitted priority	Vector offset
16	0x10 (General Illegal Instruction) 0x14 (RAM Error) 0x18 (Slot Illegal Instruction) 0x24 (CPU Address error) 0x28 (DMAC Address error) 0x2C (NMI) 0x34 (FPU exception – where applicable) 0x3C (Bank Overflow) 0x40 (Bank Underflow) 0x44 (Divide by zero) 0x48 (Divide overflow) 0x80 – 0xFC (All Software Traps)
15	0x30 (User Break) 0x38 (H-UDI)

3.1.8 Register Banking

RTA-OSEK has been configured to use register banking for all Category 2 ISRs. The register banks covered by the Category 2 priorities should be enabled before the call to `StartOS()`. If register banking is to be used for all priorities then the `__interrupt` function qualifier should result in code that uses the `resbank` assembler instruction (i.e. these functions should be compiled with the `-cpu=sh2a` option).

3.1.9 Sequential Interrupt controller workaround

If an interrupt with an interrupt priority level (IPL) equal to or lower than the current IPL triggers during an interrupt service routine (ISR) it will be serviced sequentially after the current ISR has completed. Due to the pipeline construction of the SH-2A CPU the next ISR is not entered immediately (i.e. the next instruction) after the current ISR ends. Instead a number of instructions of the interrupted code are executed between the consecutive ISRs. The number of these instructions is dependant upon the configuration of the memory and where the code is located. A workaround is provided and can be used to prevent up to four instructions from executing between successive interrupts. This has been tested at LiveDevices where the code is located in external ROM, the memory buses read and write timings are set to the maximum and the cache is not activated to set the slowest access latency. The workaround can be applied when executing code with smaller access latencies. During these tests no more than one instruction was executed between re-triggering ISRs regardless of the configuration. To use the workaround the file `osgen.sh` should be assembled with the preprocessor string literal `OS_ISR_FIX` defined. This can be achieved using the assembler command line option `-DOS_ISR_FIX`.

The workaround adds a number of instructions to the exit latency of a Category 2 ISR, the entry latency is unaffected. The workaround is only advised if consecutive interrupts occur in an application (i.e. the ISR re-triggers or there are multiple concurrently active Category 2 ISRs). It is also advised that care should be taken to ensure that any variable data objects shared between tasks and ISRs are protected with the appropriate resource locks.

The workaround assumes that when an interrupt triggers the saved context is successfully placed in a register bank. If all of the register banked have been used when an interrupt triggers the bank overflow exception must be triggered (i.e. the BOVE bit in IBNR of the INTC is 1). If the bank overflow is not triggered then the stack will be corrupted.

Important: When using the interrupt controller workaround the bank overflow exception (BOVE) must be configured.

3.2 Register Settings

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

Register	Notes
`IPRx`	Interrupt Priority Registers need to be set for all interrupt sources that use them RTA-OSEK generates suitable values in `os_ipr_values`.
`IBNR[BN]`	The use of register banks should be enabled before calling `StartOS()`.
`IBCR[E1-E15]`	All register banks should be enabled before calling `StartOS()`.
`VBR`	Vector Base Register must be set before calling `StartOS()`

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

Register	Notes
`SR[IMASK]`	The IMASK bits of the SR 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): 36

Timing

API max usage (bytes): 36

Extended

API max usage (bytes): 68

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.

Important: RTA-OSEK expects the stack to be located in the default stack section `stack`. The initial stack pointer (R15) value must be equal to the symbol automatically generated by the Green Hills linker `__ghsend_stack`.

3.4 Floating point

SH2A-FPU CPUs contain a single precision hardware floating-point arithmetic unit that is not part of the SH-2A CPU. When floating-point hardware is used for more than one task or ISR, the contents of the floating-point registers must be saved to prevent corruption of their values.

An example of how to save floating-point context can be found in `osfptgt.c` and `osfptgt.h` in the *<RTA-OSEK location>*`\ghssh2a\inc` directory. For an application to use floating-point context saving, the appropriate tasks and Category 2 interrupt service routines must be marked as using floating-point operation in the RTA-OSEK GUI.

Important: The RTA-OSEK libraries contain a pre-compiled version of `osfptgt.c` to support SH2A-FPU CPUs. If a task is marked as using floating-point context in an application run on an SH-2A CPU an exception will occur, as the floating-point instructions used in the default implementation are not supported on a CPU that does not contain floating-point hardware. In this case `osfptgt.c` should be recompiled for the SH-2A CPU.

Note that the performance figures have been collected for a system using hardware floating-point.

3.5 Counters

Different SH-2A variants provide different width (i.e. 16-bit and 32-bit) hardware counters. RTA-OSEK assumes that the hardware counter used to provide the stopwatch uses the full 32-bit width. If this is not the case then the remaining bits should be supported in software. The example application demonstrates a method how this can be done.

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	**SH2A SH72513**
Clock speed (MHz)	80
Code memory	on-chip ROM
Read-only data memory	on-chip 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	32	32	32	32	32	32
Maximum number of not suspended tasks	32	32	32	32	32	32
Maximum number of priorities	32	32	32	32	32	32
Number of tasks per priority (for BCC2 and ECC2)	n/a	32	32	n/a	32	32
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	32	32	32
Limits for the number of alarm objects (per system / per task)	not limited by RTA-OSEK					
Limits for the number of standard resources (per system)	255	255	255	255	255	255
Limits for the number of internal resources (per system)	not limited by RTA-OSEK					
Limits for the number of nested resources (per system / per task)	255	255	255	255	255	255

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Limits for the number of application modes	4294967295					

4.2 Hardware Resources

4.2.1 ROM and RAM Overheads

The following tables give the ROM and RAM overheads for the RTA-OSEK Component (in bytes). The OSEK COM overheads are quoted separately. If you do not use messages, your application will not include this overhead for the parts of OSEK COM required to implement messaging.

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	28	28	28	28	28	28
	ROM	196	196	196	196	196	196
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	268	268	268	268	268	268
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	66	66	66	66	66	66
	ROM	314	314	314	314	314	314
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

4.2.2 ROM and RAM for OSEK OS Objects

In addition to the base OS overhead, detailed in Section 4.2.1, each OSEK OS object requires ROM and/or RAM. RTA-OSEK provides additional sub-task types for each task type in OSEK (basic and extended), determined by the offline configuration tools. They are as follows:

OSEK Class	Termination	Arithmetic
BCC1	Lightweight	Integer or Floating-Point
BCC1	Heavyweight	Integer or Floating-Point
BCC2	Light or Heavy	Integer or Floating-Point
ECC1	Heavyweight	Integer
ECC1	Heavyweight	Floating-Point
ECC2	Heavyweight	Integer
ECC2	Heavyweight	Floating-Point

The following tables give the ROM and/or RAM requirements (in bytes) for each OS object in the RTA-OSEK Component. (Note that the OSEK COM class was set to CCCA for systems without events, CCCB for systems with events. A default message of size 10 bytes was used for both CCCA and CCCB. The CCCB message size includes queued messages.)

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	36	36	36	36	36	36
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	40	40	40	40	40	40
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	44	52	n/a	44	52
ECC1, Integer task	RAM	n/a	n/a	n/a	60	60	60
	ROM	n/a	n/a	n/a	60	60	60
ECC1, floating-point task	RAM	n/a	n/a	n/a	62	62	62
	ROM	n/a	n/a	n/a	60	60	60
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	62
	ROM	n/a	n/a	n/a	n/a	n/a	68
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	64
	ROM	n/a	n/a	n/a	n/a	n/a	68
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	48	48	48	48	48	48
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	78	78	78	78	78	78
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	50	50	50	50	50	50
Counter	RAM	4	4	4	4	4	4
	ROM	22	22	22	22	22	22
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	48	48	48	48	48	48
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	52	52	52	52	52	52
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	56	64	n/a	56	64
ECC1, Integer task	RAM	n/a	n/a	n/a	72	72	72
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating-point task	RAM	n/a	n/a	n/a	74	74	74
	ROM	n/a	n/a	n/a	72	72	72
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	74
	ROM	n/a	n/a	n/a	n/a	n/a	80
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	76
	ROM	n/a	n/a	n/a	n/a	n/a	80

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	110	110	110	110	110	110
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	126	126	126	126	126	126
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	50	50	50	50	50	50
Counter	RAM	4	4	4	4	4	4
	ROM	22	22	22	22	22	22
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	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	64	72	n/a	64	72
ECC1, Integer task	RAM	n/a	n/a	n/a	76	76	76
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	78	78	78
	ROM	n/a	n/a	n/a	80	80	80
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	78
	ROM	n/a	n/a	n/a	n/a	n/a	88
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	80
	ROM	n/a	n/a	n/a	n/a	n/a	88
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	122	122	122	122	122	122
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	138	138	138	138	138	138
Resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	54	54	54	54	54	54
Counter	RAM	4	4	4	4	4	4
	ROM	26	26	26	26	26	26
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	8	8	8	8	8	8
	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	128	188	236	144	200	260
	NS		104	156	208	116	164	228
	KL	2	64	112	164	72	124	188

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	24	24	24	24	24	24
ChainTask	SWL	1, 8	116	184	224	136	192	244
	SWH	1, 9	144	204	248	160	212	272
	NSL	8	116	184	224	136	192	244
	NSH	9	132	192	240	148	204	260
Schedule			100	100	132	100	100	132
GetTaskID			32	32	32	32	32	32
GetTaskState			80	80	80	96	96	96
EnableAllInterrupts			32	32	32	32	32	32
DisableAllInterrupts			24	24	24	24	24	24
ResumeAllInterrupts			48	48	48	48	48	48
SuspendAllInterrupts			40	40	40	40	40	40
ResumeOSInterrupts			40	40	40	40	40	40
SuspendOSInterrupts			60	60	60	60	60	60
GetResource	Task	7	28	28	32	28	28	32
	Combined	6	68	68	68	68	68	68
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	80	80	80	80	80	80
	Combined	6	160	160	160	160	160	160
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	124	124	212
	NS		n/a	n/a	n/a	92	92	176
	NS1i	10	n/a	n/a	n/a	52	n/a	n/a
	KL	2	n/a	n/a	n/a	60	60	144
	KL1i	2, 10	n/a	n/a	n/a	24	n/a	n/a
ClearEvent			n/a	n/a	n/a	48	48	48
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	280	280	420
	fp	11	n/a	n/a	n/a	272	272	468
	1i	10	n/a	n/a	n/a	20	n/a	n/a
GetAlarmBase			56	56	56	56	56	56
GetAlarm			104	104	104	104	104	104
SetRelAlarm			124	124	124	124	124	124
SetAbsAlarm			160	160	160	160	160	160
CancelAlarm			92	92	92	92	92	92
InitCounter			56	56	56	56	56	56

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			76	76	76	76	76	76
osek_tick_alarm	<default>		68	68	68	68	68	68
	KL	2	36	36	36	36	36	36
osek_incr_counter			44	44	44	44	44	44
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			192	192	192	192	192	192
ShutdownOS	NoHook	12	32	32	32	32	32	32
	Hook	13	48	48	48	48	48	48
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			36	36	36	36	36	36
StopCOM			20	20	20	20	20	20
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	80	80	80	216	216	216
	CCCB	15	216	216	216	216	216	216
GetMessageResource			48	48	48	48	48	48
ReleaseMessageResource			44	44	44	44	44	44
GetMessageStatus			52	52	52	52	52	52
SendMessage	SW CCCA	1, 14	128	128	128	292	292	292
	SW CCCB	1, 15	276	276	276	292	292	292
	NS CCCA	14	128	128	128	292	292	292
	NS CCCB	15	276	276	276	292	292	292
	KL CCCA	2, 14	92	92	92	260	260	260
	KL CCCB	2, 15	244	244	244	260	260	260
main_dispatch	NoHook	12	128	128	168	128	128	168
	Hook	13	172	172	204	172	172	204
sub_dispatch	B1LF	19	32	32	32	32	32	32
	B1HI	20	104	104	104	104	104	104
	B1HF	21	120	120	120	120	120	120
	B2LI	22	n/a	72	100	n/a	72	100
	B2LF	23	n/a	88	116	n/a	88	116
	B2HI	24	n/a	168	224	n/a	168	224
	B2HF	25	n/a	180	240	n/a	180	240
	E1HI	26	n/a	n/a	n/a	324	324	396
	E1HF	27	n/a	n/a	n/a	340	340	412
	E2HI	28	n/a	n/a	n/a	n/a	n/a	396

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	412
ErrorHook support		16	48	48	48	48	48	48
	ServiceID	17	56	56	56	56	56	56
	Parameters	18	68	68	68	68	68	68
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	84	140	196	96	164	224
	NS		52	108	164	64	132	192
	KL	2	20	72	128	32	96	156
ChainTaskset	SWL	1, 8	60	108	164	60	120	180
	SWH	1, 9	96	156	212	96	168	228
	NSL	8	60	108	164	60	120	180
	NSH	9	88	144	200	88	156	216
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			48	48	48	48	48	48
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			48	48	48	48	48	48
TestSubTaskset			56	56	56	56	56	56
TestEquivalentTaskset			52	52	52	52	52	52
TickSchedule	SW	1	136	140	140	140	140	140
	NS		104	104	104	104	104	104
	KL	2	68	72	72	72	72	72
AdvanceSchedule	SW	1	152	148	148	148	148	148
	NS		124	112	112	112	112	112
	KL	2	84	72	72	72	72	72
StartSchedule			92	92	92	92	92	92
StopSchedule			68	68	68	68	68	68
GetScheduleStatus			112	112	112	112	112	112
GetScheduleValue			80	80	80	80	80	80
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			8	8	8	8	8	8
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			20	20	20	20	20	20
Floating point support			208	208	208	208	208	208
Interrupt support			64	64	64	64	64	64
Heavyweight task support			78	78	78	78	78	78
Utility function			104	104	104	104	104	104

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	128	188	236	144	200	260
	NS		104	156	208	116	164	228
	KL	2	64	112	164	72	124	188
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	24	24	24	24	24	24
ChainTask	SWL	1, 8	116	184	224	136	192	244
	SWH	1, 9	144	204	248	160	212	272
	NSL	8	116	184	224	136	192	244
	NSH	9	132	192	240	148	204	260
Schedule			120	120	152	120	120	152
GetTaskID			32	32	32	32	32	32
GetTaskState			80	80	80	96	96	96
EnableAllInterrupts			32	32	32	32	32	32
DisableAllInterrupts			24	24	24	24	24	24
ResumeAllInterrupts			48	48	48	48	48	48
SuspendAllInterrupts			40	40	40	40	40	40
ResumeOSInterrupts			40	40	40	40	40	40
SuspendOSInterrupts			60	60	60	60	60	60
GetResource	Task	7	28	28	32	28	28	32

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Combined	6	68	68	68	68	68	68
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	100	100	100	100	100	100
	Combined	6	196	196	196	196	196	196
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	124	124	212
	NS		n/a	n/a	n/a	92	92	176
	NS1i	10	n/a	n/a	n/a	52	n/a	n/a
	KL	2	n/a	n/a	n/a	60	60	144
	KL1i	2, 10	n/a	n/a	n/a	24	n/a	n/a
ClearEvent			n/a	n/a	n/a	48	48	48
GetEvent			n/a	n/a	n/a	12	12	12
WaitEvent	<default>		n/a	n/a	n/a	372	372	488
	fp	11	n/a	n/a	n/a	344	344	540
	1i	10	n/a	n/a	n/a	112	n/a	n/a
GetAlarmBase			56	56	56	56	56	56
GetAlarm			104	104	104	104	104	104
SetRelAlarm			124	124	124	124	124	124
SetAbsAlarm			160	160	160	160	160	160
CancelAlarm			92	92	92	92	92	92
InitCounter			56	56	56	56	56	56
GetCounterValue			76	76	76	76	76	76
osek_tick_alarm	<default>		68	68	68	68	68	68
	KL	2	36	36	36	36	36	36
osek_incr_counter			44	44	44	44	44	44
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			248	248	248	248	248	248
ShutdownOS	NoHook	12	32	32	32	32	32	32
	Hook	13	48	48	48	48	48	48
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			36	36	36	36	36	36
StopCOM			20	20	20	20	20	20
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	80	80	80	216	216	216
	CCCB	15	216	216	216	216	216	216

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageResource			48	48	48	48	48	48
ReleaseMessageResource			44	44	44	44	44	44
GetMessageStatus			52	52	52	52	52	52
SendMessage	SW CCCA	1, 14	128	128	128	292	292	292
	SW CCCB	1, 15	276	276	276	292	292	292
	NS CCCA	14	128	128	128	292	292	292
	NS CCCB	15	276	276	276	292	292	292
	KL CCCA	2, 14	92	92	92	260	260	260
	KL CCCB	2, 15	244	244	244	260	260	260
main_dispatch	NoHook	12	172	172	212	172	172	212
	Hook	13	212	212	248	212	212	248
sub_dispatch	B1LF	19	28	28	28	28	28	28
	B1HI	20	108	108	108	108	108	108
	B1HF	21	124	124	124	124	124	124
	B2LI	22	n/a	52	80	n/a	52	80
	B2LF	23	n/a	68	96	n/a	68	96
	B2HI	24	n/a	144	208	n/a	144	208
	B2HF	25	n/a	160	224	n/a	160	224
	E1HI	26	n/a	n/a	n/a	364	364	440
	E1HF	27	n/a	n/a	n/a	380	380	456
	E2HI	28	n/a	n/a	n/a	n/a	n/a	440
	E2HF	29	n/a	n/a	n/a	n/a	n/a	456
ErrorHook support		16	48	48	48	48	48	48
	ServiceID	17	56	56	56	56	56	56
	Parameters	18	68	68	68	68	68	68
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	84	84	84	84	84	84
Timing_termination		4	80	80	80	80	80	80
ActivateTaskset	SW	1	84	140	196	96	164	224
	NS		52	108	164	64	132	192
	KL	2	20	72	128	32	96	156
ChainTaskset	SWL	1, 8	60	108	164	60	120	180
	SWH	1, 9	96	156	212	96	168	228
	NSL	8	60	108	164	60	120	180
	NSH	9	88	144	200	88	156	216
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			48	48	48	48	48	48

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			48	48	48	48	48	48
TestSubTaskset			56	56	56	56	56	56
TestEquivalentTaskset			52	52	52	52	52	52
TickSchedule	SW	1	136	140	140	140	140	140
	NS		104	104	104	104	104	104
	KL	2	68	72	72	72	72	72
AdvanceSchedule	SW	1	152	148	148	148	148	148
	NS		124	112	112	112	112	112
	KL	2	84	72	72	72	72	72
StartSchedule			92	92	92	92	92	92
StopSchedule			68	68	68	68	68	68
GetScheduleStatus			112	112	112	112	112	112
GetScheduleValue			80	80	80	80	80	80
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			8	8	8	8	8	8
SetArrivalpointNext			8	8	8	8	8	8
TestArrivalpointWritable			32	32	32	32	32	32
GetExecutionTime			108	108	108	108	108	108
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			12	12	12	12	12	12
GetStackOffset			20	20	20	20	20	20
Floating point support			208	208	208	208	208	208
Interrupt support			236	236	236	236	236	236
Heavyweight task support			78	78	78	78	78	78
Utility function			104	104	104	104	104	104

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	280	340	376	296	348	400
	NS		332	392	428	352	400	468
	KL	2	180	240	276	196	248	300
TerminateTask	LExt	3	156	156	156	156	156	156
	H	5	148	148	148	148	148	148
ChainTask	SWL	1, 8	292	356	396	304	368	416
	SWH	1, 9	340	396	436	352	408	456
	NSL	8	360	424	460	372	432	484
	NSH	9	384	440	480	396	452	500
Schedule			264	264	296	264	264	296
GetTaskID			48	48	48	48	48	48
GetTaskState			296	296	296	288	288	288
EnableAllInterrupts			48	48	48	48	48	48
DisableAllInterrupts			40	40	40	40	40	40
ResumeAllInterrupts			104	104	104	104	104	104
SuspendAllInterrupts			56	56	56	56	56	56
ResumeOSInterrupts			96	96	96	96	96	96
SuspendOSInterrupts			76	76	76	76	76	76
GetResource	Task	7	344	344	304	344	344	304
	Combined	6	424	424	424	424	424	424
	CLEx	3	368	368	368	368	368	368
ReleaseResource	Task	7	340	340	340	340	340	340
	Combined	6	432	432	432	432	432	432
	CLEx	3	288	288	288	288	288	288
SetEvent	SW	1	n/a	n/a	n/a	328	328	404
	NS		n/a	n/a	n/a	364	364	440
	NS1i	10	n/a	n/a	n/a	260	n/a	n/a
	KL	2	n/a	n/a	n/a	236	236	312
	KL1i	2, 10	n/a	n/a	n/a	184	n/a	n/a
ClearEvent			n/a	n/a	n/a	140	140	140
GetEvent			n/a	n/a	n/a	188	188	188
WaitEvent	<default>		n/a	n/a	n/a	436	436	556
	fp	11	n/a	n/a	n/a	464	464	608

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	232	n/a	n/a
GetAlarmBase			212	212	212	212	212	212
GetAlarm			220	220	220	220	220	220
SetRelAlarm			284	284	284	284	284	284
SetAbsAlarm			304	304	304	304	304	304
CancelAlarm			204	204	204	204	204	204
InitCounter			284	284	284	284	284	284
GetCounterValue			224	224	224	224	224	224
osek_tick_alarm	<default>		140	140	140	140	140	140
	KL	2	36	36	36	36	36	36
osek_incr_counter			44	44	44	44	44	44
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			264	264	264	264	264	264
ShutdownOS	NoHook	12	44	44	44	44	44	44
	Hook	13	56	56	56	56	56	56
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			52	52	52	52	52	52
StopCOM			52	52	52	52	52	52
ReadFlag			28	28	28	28	28	28
ResetFlag			36	36	36	36	36	36
ReceiveMessage	CCCA	14	208	208	208	324	324	324
	CCCB	15	324	324	324	324	324	324
GetMessageResource			104	104	104	104	104	104
ReleaseMessageResource			108	108	108	108	108	108
GetMessageStatus			124	124	124	124	124	124
SendMessage	SW CCCA	1, 14	260	260	260	432	432	432
	SW CCCB	1, 15	416	416	416	432	432	432
	NS CCCA	14	260	260	260	432	432	432
	NS CCCB	15	416	416	416	432	432	432
	KL CCCA	2, 14	188	188	188	352	352	352
	KL CCCB	2, 15	336	336	336	352	352	352
main_dispatch	NoHook	12	172	172	212	172	172	212
	Hook	13	212	212	248	212	212	248
sub_dispatch	B1LF	19	28	28	28	28	28	28
	B1HI	20	108	108	108	108	108	108
	B1HF	21	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	
	B2LI	22	n/a	52	80	n/a	52	80
	B2LF	23	n/a	68	96	n/a	68	96
	B2HI	24	n/a	144	208	n/a	144	208
	B2HF	25	n/a	160	224	n/a	160	224
	E1HI	26	n/a	n/a	n/a	364	364	440
	E1HF	27	n/a	n/a	n/a	380	380	456
	E2HI	28	n/a	n/a	n/a	n/a	n/a	440
	E2HF	29	n/a	n/a	n/a	n/a	n/a	456
ErrorHook support		16	132	132	132	132	132	132
	ServiceID	17	140	140	140	140	140	140
	Parameters	18	156	156	156	156	156	156
validity_checks		3	28	28	28	28	28	28
Timing_dispatch		4	84	84	84	84	84	84
Timing_termination		4	80	80	80	80	80	80
ActivateTaskset	SW	1	324	392	440	364	420	484
	NS		364	440	484	396	468	528
	KL	2	212	300	344	260	328	372
ChainTaskset	SWL	1, 8	352	436	480	388	456	524
	SWH	1, 9	396	484	528	444	508	568
	NSL	8	408	500	544	452	520	576
	NSH	9	452	536	580	492	560	616
GetTasksetRef			152	152	152	152	152	152
MergeTaskset			264	264	264	264	264	264
AssignTaskset			148	148	148	148	148	148
RemoveTaskset			268	268	268	268	268	268
TestSubTaskset			276	276	276	276	276	276
TestEquivalentTaskset			276	276	276	276	276	276
TickSchedule	SW	1	356	288	288	288	288	288
	NS		428	348	348	348	348	348
	KL	2	256	176	176	176	176	176
AdvanceSchedule	SW	1	364	284	284	284	284	284
	NS		412	356	356	356	356	356
	KL	2	284	204	204	204	204	204
StartSchedule			284	284	284	284	284	284
StopSchedule			208	208	208	208	208	208
GetScheduleStatus			260	260	260	260	260	260
GetScheduleValue			220	220	220	220	220	220

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleNext			84	84	84	84	84	84
SetScheduleNext			164	164	164	164	164	164
GetArrivalpointDelay			128	128	128	128	128	128
SetArrivalpointDelay			144	144	144	144	144	144
GetArrivalpointTasksetRef			128	128	128	128	128	128
GetArrivalpointNext			128	128	128	128	128	128
SetArrivalpointNext			172	172	172	172	172	172
TestArrivalpointWritable			152	152	152	152	152	152
GetExecutionTime			172	172	172	172	172	172
GetLargestExecutionTime			120	120	120	120	120	120
ResetLargestExecutionTime			112	112	112	112	112	112
GetStackOffset			20	20	20	20	20	20
Floating point support			208	208	208	208	208	208
Interrupt support			236	236	236	236	236	236
Heavyweight task support			78	78	78	78	78	78
Utility function			104	104	104	104	104	104

Notes

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

Number	Note
16	Linked only if `USEGETSERVICEID = FALSE` and `USEPARAMETERACCESS = FALSE`
17	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = FALSE`
18	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = TRUE`
19	Linked only for basic, single-activation, lightweight, floating-point tasks
20	Linked only for basic, single-activation, heavyweight, integer tasks
21	Linked only for basic, single-activation, heavyweight, floating-point tasks
22	Linked only for basic, multiple-activation, lightweight, integer tasks
23	Linked only for basic, multiple-activation, lightweight, floating-point tasks
24	Linked only for basic, multiple-activation, heavyweight, integer tasks
25	Linked only for basic, multiple-activation, heavyweight, floating-point tasks
26	Linked only for extended, unique priority, integer tasks
27	Linked only for extended, unique priority, floating-point tasks
28	Linked only for extended, shared priority, integer tasks
29	Linked only for extended, shared priority, floating-point tasks
30	Implemented as a macro, so no code is linked
31	Not required on some targets

4.2.4 Reserved Hardware Resources

Timer units, interrupts, traps and other hardware resources are not reserved by RTA-OSEK.

4.3 Performance

The collection of performance data for the SH2A/GreenHills port of the RTA-OSEK Component was achieved using a timer running four times slower than the CPU clock speed. The figures in this section, therefore, have an uncertainty level of up to four CPU cycles. The actual times are between 0 and four cycles shorter than those reported in the remainder of this section.

4.3.1 Execution Times for RTA-OSEK API Calls

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	84	124	156	92	120	160
	NS	76	112	148	72	104	144
	KL	52	84	120	56	84	132
TerminateTask	LExt	0	0	0	0	0	0
	H	160	164	168	160	164	164
ChainTask	SWL	296	344	396	368	392	460
	SWH	340	392	448	412	444	504
	NSL	292	340	396	368	388	460
	NSH	336	384	444	404	440	500
Schedule	SW	92	92	88	92	88	88
GetTaskID		48	48	48	44	44	44
GetTaskState		68	72	68	80	80	80
EnableAllInterrupts		36	36	36	36	36	36
DisableAllInterrupts		32	32	32	32	32	32
ResumeAllInterrupts		44	44	44	44	44	44
SuspendAllInterrupts		44	44	44	44	44	44
ResumeOSInterrupts		44	44	44	44	44	44
SuspendOSInterrupts		44	44	44	40	44	40
GetResource	Task	44	48	44	48	44	44
	Combined	52	56	52	56	52	52
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	72	76	76	76	72	72
	Combined	100	104	108	104	100	100
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	104	104	124
	NS	n/a	n/a	n/a	76	76	100
	KL	n/a	n/a	n/a	64	64	72
ClearEvent		n/a	n/a	n/a	52	52	52
GetEvent		n/a	n/a	n/a	28	32	32
WaitEvent	<default>	n/a	n/a	n/a	464	468	508
	fp	n/a	n/a	n/a	448	452	512
GetAlarmBase		68	68	68	68	68	68
GetAlarm		84	84	88	84	84	84

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		100	96	100	96	96	100
SetAbsAlarm		108	108	104	104	104	104
CancelAlarm		76	76	76	80	80	76
InitCounter		60	56	56	56	56	56
GetCounterValue		60	56	56	56	56	56
osek_tick_alarm	<default>	64	64	68	64	64	68
	KL	44	44	44	44	44	44
osek_incr_counter		24	24	24	24	24	24
GetActiveApplicationMode		28	28	28	28	28	28
StartOS		804	804	804	804	808	804
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	48	48	48	48	48	48
InitCOM		24	24	24	24	24	24
CloseCOM		24	24	24	24	24	24
StartCOM		60	60	60	140	136	136
StopCOM		32	32	32	32	32	32
ReadFlag		n/a	n/a	n/a	24	24	24
ResetFlag		n/a	n/a	n/a	24	24	24
ReceiveMessage		44	44	44	208	212	212
GetMessageResource		n/a	n/a	n/a	100	100	96
ReleaseMessageResource		n/a	n/a	n/a	128	124	128
GetMessageStatus		n/a	n/a	n/a	56	56	56
SendMessage	SW	172	216	244	348	372	412
	NS	168	204	232	324	356	396
	KL	116	148	184	284	312	356
ActivateTaskset	SW	68	516	508	76	508	512
	NS	52	496	476	56	492	488
	KL	32	468	456	36	468	472
	SW2	68	516	508	76	508	512
	NS2	52	496	476	56	492	488
	KL2	32	468	456	36	468	472
ChainTaskset	SWL	280	732	740	340	776	800
	SWH	344	796	800	404	840	868
	NSL	280	728	732	344	780	800
	NSH	336	792	796	400	836	856
GetTasksetRef		32	32	32	32	32	32

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		52	56	52	52	52	56
AssignTaskset		28	28	28	28	28	28
RemoveTaskset		52	52	52	52	52	52
TestSubTaskset		60	60	60	60	64	60
TestEquivalentTaskset		60	60	60	60	56	60
TickSchedule	SW	128	588	576	156	600	600
	NS	104	572	560	136	580	584
	KL	80	548	536	108	552	560
	SW2	128	588	576	152	588	592
	NS2	104	572	560	132	568	576
	KL2	80	548	536	104	540	552
AdvanceSchedule	SW	144	584	572	152	600	600
	NS	124	572	556	136	584	584
	KL	104	544	528	104	552	552
	SW2	144	584	572	152	588	592
	NS2	124	572	556	136	572	576
	KL2	104	540	528	104	540	544
StartSchedule		100	104	104	100	100	104
StopSchedule		88	88	88	88	84	88
GetScheduleStatus		100	100	100	96	100	100
GetScheduleValue		100	96	100	96	96	100
GetScheduleNext		32	32	32	36	36	36
SetScheduleNext		32	28	28	28	32	32
GetArrivalpointDelay		28	28	28	28	28	28
SetArrivalpointDelay		28	28	28	28	28	28
GetArrivalpointTasksetRef		24	24	24	24	24	24
GetArrivalpointNext		28	28	28	28	28	24
SetArrivalpointNext		28	28	28	28	28	28
TestArrivalpointWritable		36	36	36	36	32	32
GetExecutionTime		20	20	20	20	20	20
GetLargestExecutionTime		28	28	28	28	28	28
ResetLargestExecutionTime		20	20	20	20	20	20
GetStackOffset		32	32	32	32	32	36

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	80	116	160	92	120	160
	NS	72	108	144	72	112	144
	KL	52	84	120	56	80	132
TerminateTask	LExt	0	0	0	0	0	0
	H	372	376	380	368	376	372
ChainTask	SWL	544	580	640	608	636	708
	SWH	592	628	692	648	688	752
	NSL	544	576	640	608	636	708
	NSH	584	620	684	644	680	752
Schedule	SW	92	92	88	88	92	88
GetTaskID		48	48	48	44	44	44
GetTaskState		72	72	72	80	80	80
EnableAllInterrupts		36	36	36	36	36	36
DisableAllInterrupts		32	32	32	32	32	32
ResumeAllInterrupts		44	44	44	44	44	44
SuspendAllInterrupts		44	44	44	44	44	44
ResumeOSInterrupts		44	44	44	44	44	44
SuspendOSInterrupts		44	44	44	44	44	40
GetResource	Task	44	48	48	44	44	48
	Combined	52	56	56	52	52	56
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	72	76	72	72	72	72
	Combined	100	104	100	100	100	100
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	108	104	128
	NS	n/a	n/a	n/a	80	80	100
	KL	n/a	n/a	n/a	68	64	76
ClearEvent		n/a	n/a	n/a	52	52	52
GetEvent		n/a	n/a	n/a	28	32	28
WaitEvent	<default>	n/a	n/a	n/a	664	660	708
	fp	n/a	n/a	n/a	656	656	716
GetAlarmBase		68	68	68	68	68	68
GetAlarm		88	84	84	84	84	84

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		96	96	96	100	96	100
SetAbsAlarm		104	104	104	108	108	104
CancelAlarm		80	80	80	76	76	76
InitCounter		56	56	56	56	56	60
GetCounterValue		56	56	56	56	56	60
osek_tick_alarm	<default>	64	64	64	64	64	68
	KL	44	44	44	44	44	44
osek_incr_counter		24	24	24	24	24	24
GetActiveApplicationMode		28	28	28	28	28	28
StartOS		1924	1920	1928	1924	1924	1932
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	48	48	48	48	48	48
InitCOM		24	24	24	24	24	24
CloseCOM		24	24	24	24	24	24
StartCOM		56	60	60	140	140	140
StopCOM		32	32	32	32	32	32
ReadFlag		n/a	n/a	n/a	24	24	24
ResetFlag		n/a	n/a	n/a	24	24	24
ReceiveMessage		48	44	44	212	212	208
GetMessageResource		n/a	n/a	n/a	100	100	100
ReleaseMessageResource		n/a	n/a	n/a	128	128	128
GetMessageStatus		n/a	n/a	n/a	52	56	56
SendMessage	SW	168	208	248	344	372	416
	NS	164	200	236	328	360	396
	KL	116	156	188	280	312	356
ActivateTaskset	SW	72	516	496	76	508	512
	NS	52	492	472	56	488	488
	KL	32	472	460	36	468	472
	SW2	72	516	496	76	508	512
	NS2	52	492	472	56	488	488
	KL2	32	472	460	36	468	472
ChainTaskset	SWL	532	968	976	588	1024	1048
	SWH	596	1028	1048	640	1088	1112
	NSL	528	964	976	584	1024	1044
	NSH	584	1024	1036	632	1084	1108
GetTasksetRef		32	32	32	32	32	32

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		52	56	56	52	52	56
AssignTaskset		28	28	28	28	28	28
RemoveTaskset		52	52	52	52	52	52
TestSubTaskset		64	64	64	64	60	60
TestEquivalentTaskset		56	56	56	56	60	60
TickSchedule	SW	128	592	580	152	600	600
	NS	108	572	560	132	580	580
	KL	80	548	536	108	552	556
	SW2	128	592	580	152	588	592
	NS2	108	572	560	132	568	572
	KL2	80	548	536	108	540	548
AdvanceSchedule	SW	144	588	576	152	600	596
	NS	124	572	560	132	584	580
	KL	104	544	532	100	552	552
	SW2	144	588	576	152	588	588
	NS2	120	572	560	132	572	572
	KL2	104	544	532	100	540	544
StartSchedule		100	100	100	104	100	104
StopSchedule		88	88	88	88	88	88
GetScheduleStatus		100	100	100	100	100	100
GetScheduleValue		96	96	96	100	96	100
GetScheduleNext		32	32	32	36	36	36
SetScheduleNext		32	32	32	28	32	32
GetArrivalpointDelay		28	28	28	28	28	28
SetArrivalpointDelay		28	28	28	28	28	28
GetArrivalpointTasksetRef		24	24	28	24	24	24
GetArrivalpointNext		24	28	24	24	28	24
SetArrivalpointNext		28	28	28	28	28	28
TestArrivalpointWritable		32	36	36	32	32	36
GetExecutionTime		128	128	132	132	136	132
GetLargestExecutionTime		40	40	40	40	36	36
ResetLargestExecutionTime		32	28	32	28	32	28
GetStackOffset		32	32	32	32	32	36

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	248	280	304	248	276	308
	NS	272	308	336	284	312	348
	KL	208	248	268	216	244	280
TerminateTask	LExt	436	444	440	444	452	440
	H	484	480	484	484	484	484
ChainTask	SWL	780	828	876	864	884	952
	SWH	832	872	928	908	936	988
	NSL	812	864	904	892	920	972
	NSH	856	900	956	936	964	1016
Schedule	SW	152	152	160	148	152	160
GetTaskID		56	56	56	56	56	56
GetTaskState		264	264	268	272	272	272
EnableAllInterrupts		44	44	44	44	44	44
DisableAllInterrupts		40	40	40	40	40	40
ResumeAllInterrupts		68	68	68	68	68	68
SuspendAllInterrupts		56	52	56	52	56	56
ResumeOSInterrupts		68	68	68	68	72	68
SuspendOSInterrupts		56	52	56	52	56	56
GetResource	Task	404	408	252	440	432	284
	Combined	248	248	248	276	276	276
	CLEx	252	252	252	280	284	280
ReleaseResource	Task	256	252	248	284	284	284
	Combined	244	244	248	276	272	276
	CLEx	228	228	224	260	256	260
SetEvent	SW	n/a	n/a	n/a	296	296	292
	NS	n/a	n/a	n/a	300	292	300
	KL	n/a	n/a	n/a	252	256	260
ClearEvent		n/a	n/a	n/a	100	96	96
GetEvent		n/a	n/a	n/a	236	228	232
WaitEvent	<default>	n/a	n/a	n/a	760	768	808
	fp	n/a	n/a	n/a	768	776	812
GetAlarmBase		212	216	212	212	216	212
GetAlarm		216	220	216	216	216	216

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		244	244	240	244	236	244
SetAbsAlarm		236	236	236	236	232	236
CancelAlarm		192	200	196	192	200	192
InitCounter		288	288	284	288	284	288
GetCounterValue		196	196	196	196	196	196
osek_tick_alarm	<default>	108	112	108	108	112	108
	KL	44	44	44	44	44	44
osek_incr_counter		24	24	24	24	24	24
GetActiveApplicationMode		28	28	28	28	28	28
StartOS		2004	2004	2008	2004	2008	2004
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	52	52	52	52	56	52
InitCOM		24	24	24	24	24	24
CloseCOM		24	24	24	24	24	24
StartCOM		80	84	80	160	160	164
StopCOM		52	52	48	48	48	48
ReadFlag		n/a	n/a	n/a	44	44	44
ResetFlag		n/a	n/a	n/a	44	44	44
ReceiveMessage		168	168	164	312	312	312
GetMessageResource		n/a	n/a	n/a	412	408	408
ReleaseMessageResource		n/a	n/a	n/a	384	384	384
GetMessageStatus		n/a	n/a	n/a	140	140	140
SendMessage	SW	432	460	484	596	608	652
	NS	452	484	512	624	648	688
	KL	360	392	412	524	552	592
ActivateTaskset	SW	480	856	880	540	888	900
	NS	500	888	916	564	916	936
	KL	428	736	764	432	860	888
	SW2	480	856	880	540	888	900
	NS2	500	888	916	564	916	936
	KL2	428	736	764	432	860	888
ChainTaskset	SWL	1028	1344	1400	1088	1528	1508
	SWH	1076	1468	1532	1128	1536	1580
	NSL	1056	1472	1536	1120	1472	1572
	NSH	1100	1504	1572	1188	1576	1612
GetTasksetRef		204	204	204	204	208	204

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		148	148	148	148	148	148
AssignTaskset		88	92	92	92	92	92
RemoveTaskset		148	144	144	148	148	148
TestSubTaskset		152	152	152	152	152	152
TestEquivalentTaskset		152	152	148	148	148	148
TickSchedule	SW	204	900	928	596	1052	1068
	NS	248	928	964	620	1076	1092
	KL	172	856	888	552	1008	1020
	SW2	204	900	924	596	1032	1052
	NS2	248	924	960	620	1056	1080
	KL2	172	856	884	552	988	1008
AdvanceSchedule	SW	228	900	932	596	1052	1068
	NS	252	936	960	620	1084	1092
	KL	188	864	892	560	1012	1028
	SW2	228	900	928	596	1032	1056
	NS2	252	932	960	620	1064	1080
	KL2	188	860	888	560	992	1016
StartSchedule		164	164	164	160	164	160
StopSchedule		132	128	128	128	128	128
GetScheduleStatus		152	152	152	152	152	152
GetScheduleValue		148	148	144	148	148	148
GetScheduleNext		72	72	72	72	72	72
SetScheduleNext		104	104	104	104	104	104
GetArrivalpointDelay		80	80	80	80	80	80
SetArrivalpointDelay		100	100	100	100	100	100
GetArrivalpointTasksetRef		72	72	72	76	76	76
GetArrivalpointNext		76	76	76	76	76	76
SetArrivalpointNext		108	108	108	108	104	108
TestArrivalpointWritable		80	80	84	80	84	84
GetExecutionTime		164	164	168	164	168	164
GetLargestExecutionTime		192	192	192	196	196	196
ResetLargestExecutionTime		176	176	172	176	176	180
GetStackOffset		36	36	32	32	32	32

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	39	39	39	39	39	39
	Cat 2	59	91	91	91	91	91

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	39	39	39	39	39	39
	Cat 2	247	267	267	267	267	267

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	39	39	39	39	39	39
	Cat 2	247	271	271	267	267	267

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.

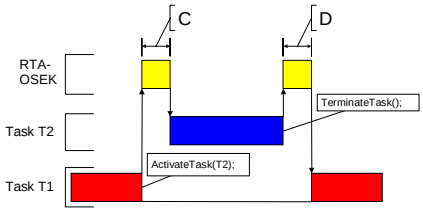

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

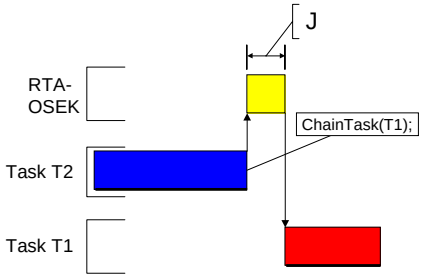

Figure 2: Task Chaining

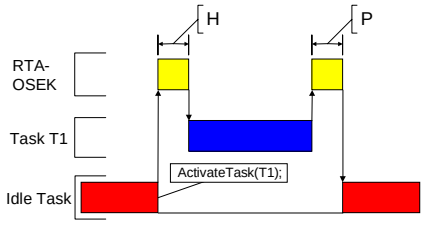

Figure 3: Task Activation from Idle Task

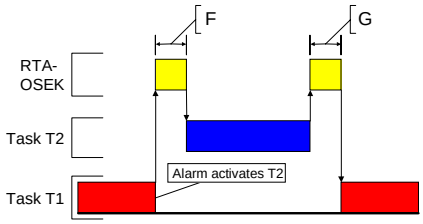

Figure 4: Task Activation from an Alarm

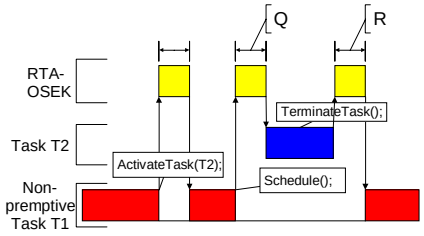

Figure 5: Non-Premptive Task Calls Schedule()

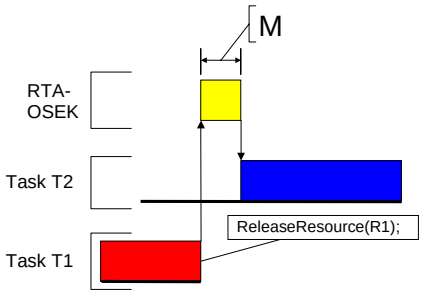

Figure 6: Blocked Task Activated by ReleaseResource()

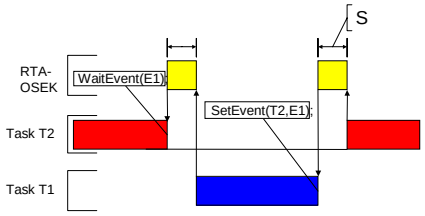

Figure 7: Waiting Task Activated by SetEvent()

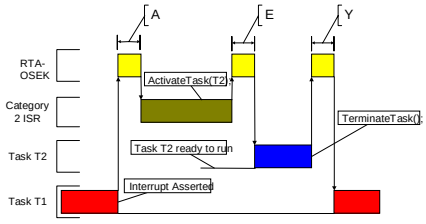

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	96	136	168	96	136	168
Figure 1: D	Heavy, Basic/Extended	156	184	208	200	200	224
ChainTask	Light, Basic	248	300	356	260	300	364
Figure 2: J	Heavy, Basic/Extended	484	568	652	536	580	664
Pre-emption	Light, Basic	196	252	332	204	248	332
Figure 1: C	Heavy, Basic/Extended	288	336	412	360	388	468
From idle task	Light, Basic	188	248	324	200	244	328
Figure 3: H	Heavy, Basic/Extended	280	332	408	352	380	460
Triggered by alarm	Light, Basic	268	324	404	280	324	408
Figure 4: F	Heavy, Basic/Extended	360	412	492	432	460	540
Schedule	Light, Basic	188	208	256	188	208	252
Figure 5: Q	Heavy, Basic/Extended	280	292	340	344	344	388
Release resource	Light, Basic	192	212	260	196	208	260
Figure 6: M	Heavy, Basic/Extended	284	296	340	348	348	396
SetEvent							

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	500	500	628
From category 2 ISR	Light, Basic	240	288	336	272	288	332
Figure 8: E	Heavy, Basic/Extended	332	372	416	424	428	468

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	312	344	372	308	348	368
Figure 1: D	Heavy, Basic/Extended	368	380	416	396	400	424
ChainTask	Light, Basic	496	548	608	508	544	612
Figure 2: J	Heavy, Basic/Extended	944	1016	1112	976	1024	1112
Pre-emption	Light, Basic	340	388	464	344	384	468
Figure 1: C	Heavy, Basic/Extended	432	464	552	488	528	612
From idle task	Light, Basic	332	380	460	340	380	460
Figure 3: H	Heavy, Basic/Extended	424	456	544	484	524	604
Triggered by alarm	Light, Basic	412	460	536	416	456	540
Figure 4: F	Heavy, Basic/Extended	504	536	624	560	600	684
Schedule	Light, Basic	332	348	392	332	344	388
Figure 5: Q	Heavy, Basic/Extended	424	424	476	476	488	532
Release resource	Light, Basic	332	352	396	336	344	392
Figure 6: M	Heavy, Basic/Extended	424	428	480	480	488	536
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	600	600	740
From category 2 ISR	Light, Basic	604	640	692	624	640	684
Figure 8: E	Heavy, Basic/Extended	688	716	772	768	780	828

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	420	460	480	428	464	476
Figure 1: D	Heavy, Basic/Extended	480	504	520	504	508	532
ChainTask	Light, Basic	736	784	836	744	792	852
Figure 2: J	Heavy, Basic/Extended	1300	1360	1440	1324	1376	1448
Pre-emption	Light, Basic	492	536	608	492	536	616
Figure 1: C	Heavy, Basic/Extended	584	628	692	656	680	760
From idle task	Light, Basic	488	532	600	484	532	608
Figure 3: H	Heavy, Basic/Extended	580	620	688	648	672	752
Triggered by alarm	Light, Basic	612	652	724	608	656	728
Figure 4: F	Heavy, Basic/Extended	700	744	808	772	796	872
Schedule	Light, Basic	384	396	456	376	396	456
Figure 5: Q	Heavy, Basic/Extended	476	484	540	540	540	600
Release resource	Light, Basic	472	484	536	496	516	560
Figure 6: M	Heavy, Basic/Extended	564	572	620	660	660	704
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	824	816	928
From category 2 ISR	Light, Basic	640	672	728	656	680	720
Figure 8: E	Heavy, Basic/Extended	728	760	808	824	820	868

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	76	76	84	76	76	84
BCC1 lightweight, floating-point	80	80	88	80	80	88
BCC1 heavyweight, integer	132	132	140	132	132	140
BCC1 heavyweight, floating-point	132	132	140	132	132	140
BCC2 lightweight, integer	n/a	80	88	n/a	80	88
BCC2 lightweight, floating-point	n/a	80	88	n/a	80	88
BCC2 heavyweight, integer	n/a	140	148	n/a	140	148
BCC2 heavyweight, floating-point	n/a	140	148	n/a	140	148
ECC1 heavyweight, integer	n/a	n/a	n/a	172	172	180
ECC1 heavyweight, floating-point	n/a	n/a	n/a	172	172	180
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	180
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	180
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	80	80	80	80	80	80
BCC1 lightweight, floating-point	84	84	84	84	84	84
BCC1 heavyweight, integer	136	136	136	136	136	136
BCC1 heavyweight, floating-point	136	136	136	136	136	136
BCC2 lightweight, integer	n/a	84	84	n/a	84	84
BCC2 lightweight, floating-point	n/a	84	84	n/a	84	84
BCC2 heavyweight, integer	n/a	144	144	n/a	144	144
BCC2 heavyweight, floating-point	n/a	144	144	n/a	144	144
ECC1 heavyweight, integer	n/a	n/a	n/a	176	176	176
ECC1 heavyweight, floating-point	n/a	n/a	n/a	176	176	176
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	176
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	176

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	80	80	92	80	80	92
BCC1 lightweight, floating-point	84	84	96	84	84	96
BCC1 heavyweight, integer	132	132	144	132	132	144
BCC1 heavyweight, floating-point	132	132	144	132	132	144
BCC2 lightweight, integer	n/a	84	96	n/a	84	96
BCC2 lightweight, floating-point	n/a	84	96	n/a	84	96
BCC2 heavyweight, integer	n/a	136	148	n/a	136	148
BCC2 heavyweight, floating-point	n/a	136	148	n/a	136	148
ECC1 heavyweight, integer	n/a	n/a	n/a	172	172	184
ECC1 heavyweight, floating-point	n/a	n/a	n/a	172	172	184
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	188
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	188
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	84	84	88	84	84	88
BCC1 lightweight, floating-point	88	88	92	88	88	92
BCC1 heavyweight, integer	136	136	140	136	136	140
BCC1 heavyweight, floating-point	136	136	140	136	136	140
BCC2 lightweight, integer	n/a	88	92	n/a	88	92
BCC2 lightweight, floating-point	n/a	88	92	n/a	88	92
BCC2 heavyweight, integer	n/a	140	144	n/a	140	144
BCC2 heavyweight, floating-point	n/a	140	144	n/a	140	144
ECC1 heavyweight, integer	n/a	n/a	n/a	176	176	180
ECC1 heavyweight, floating-point	n/a	n/a	n/a	176	176	180
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	184
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	184

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	80	80	92	80	80	92
BCC1 lightweight, floating-point	84	84	96	84	84	96
BCC1 heavyweight, integer	132	132	144	132	132	144
BCC1 heavyweight, floating-point	132	132	144	132	132	144
BCC2 lightweight, integer	n/a	84	96	n/a	84	96
BCC2 lightweight, floating-point	n/a	84	96	n/a	84	96
BCC2 heavyweight, integer	n/a	136	148	n/a	136	148
BCC2 heavyweight, floating-point	n/a	136	148	n/a	136	148
ECC1 heavyweight, integer	n/a	n/a	n/a	180	180	192
ECC1 heavyweight, floating-point	n/a	n/a	n/a	180	180	192
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	196
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	196
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	84	84	88	84	84	88
BCC1 lightweight, floating-point	88	88	92	88	88	92
BCC1 heavyweight, integer	136	136	140	136	136	140
BCC1 heavyweight, floating-point	136	136	140	136	136	140
BCC2 lightweight, integer	n/a	88	92	n/a	88	92
BCC2 lightweight, floating-point	n/a	88	92	n/a	88	92
BCC2 heavyweight, integer	n/a	140	144	n/a	140	144
BCC2 heavyweight, floating-point	n/a	140	144	n/a	140	144
ECC1 heavyweight, integer	n/a	n/a	n/a	184	184	188
ECC1 heavyweight, floating-point	n/a	n/a	n/a	184	184	188
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	192
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	192

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.