RTA-OSEK

Binding Manual: SH2A/WindRiver

Contact Details

ETAS Group www.etasgroup.com	
ETAS GmbH 70469 Stuttgart, Germany Tel.:+49 711 89661-0 Fax:+49 711 89661-300 sales.de@etas.com	**ETAS Inc.** Ann Arbor, MI 48103, USA Tel.: +1 888 ETAS INC Fax: +1 734 997 9449 sales.us@etas.com
ETAS K.K. Yokohama 220-6217, Japan Tel.: +81 45 222-0900 Fax: +81 45 222-0956 sales.jp@etas.com	**ETAS S.A.S.** 94588 Rungis Cedex, France Tel.: +33 (1) 56 70 00 50 Fax: +33 (1) 56 70 00 51 sales.fr@etas.com
ETAS Korea Co. Ltd. Seoul 137-889, Korea Tel.: +82 2 5747-016 Fax: +82 2 5747-120 sales.kr@etas.com	**ETAS Ltd.** Burton-upon-Trent Staffordshire DE14 2WQ, UK Tel.: +44 1283 54 65 12 Fax: +44 1283 54 87 67 sales.uk@etas.com
ETAS (Shanghai) Co., Ltd. Shanghai 200120, P.R. China Tel.: +86 21 5037 2220 Fax: +86 21 5037 2221 sales.cn@etas.com	**ETAS Automotive India Pvt. Ltd.** Bangalore 560 068, India Tel.: +91 80 4191 2585 Fax: +91 80 4191 2586 sales.in@etas.com

Copyright Notice

Version: M00094-002

Disclaimer

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

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

Trademarks

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

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

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

Contents

1 About this Guide

This guide provides target-specific information for the SH2A/WindRiver port of LiveDevices' RTA-OSEK. It supplements the more general information in the *RTA-OSEK User Guide*.

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

1.1 Who Should Read this Guide?

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

1.2 Conventions

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

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

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

2 Toolchain Issues

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

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

2.1 Compiler

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

Vendor	Wind River Systems, Inc.
Compiler	Wind River (Diab) Compiler for SuperH
Version	5.6.1.0-3

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

Option	Description
`-t%CPU_TYPE%`	Select the correct target for code generation

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

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

Option	Description
`-c`	Stop after the assembly step and produce an object file with default file extension .o.
`-g0`	Do not generate symbolic debugger information
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.
`-XO`	Enable all standard optimizations

To support the use of multiple CPU configurations the environment variable CPU_TYPE should be set up to match the desired CPU target (e.g. SH2AEH). `CPU_TYPE` should be set to be equal to a –t*tof* where *t* is the target processor SH2A, *o* is object format E for ELF and *f* is the floating point support where H is for hardware floating point as demonstrated in the RTA-OSEK example application.

2.2 Assembler

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

Vendor	Wind River Systems, Inc.
Assembler	Wind River Systems, Inc.
Version	5.6.1.0-3

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

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

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

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

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

2.3 Linker/Locator

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

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

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

Sections	ROM/RAM	Description
`os_pid`	ROM	RTA-OSEK read-only data
`os_pird`	ROM	RTA-OSEK initialization data
`os_text`	ROM	RTA-OSEK code
`os_vectbl`	ROM	Vector table if generated by RTA-OSEK GUI
`os_pir`	RAM	RTA-OSEK initialized data; initialized during StartOS
`os_pir2`	RAM	RTA-OSEK initialized data; must be initialized during C-startup
`os_pur`	RAM	RTA-OSEK uninitialized data; zeroed during StartOS
`os_pur2`	RAM	RTA-OSEK uninitialized data; must be zeroed during C-startup
`os_trace_ram`	RAM	RTA-TRACE uninitialized data; must be zeroed during C-startup
`os_cntr`	RAM	RTA-OSEK counter data; must be zeroed during C-startup.

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
`os_vectbl`	The RTA-OSEK vector table does not include values for the first four entries as these contained reset values for the PC and SP. The table should be located with an offset of 16 bytes from the value contained in the VBR register.

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 (build 13751-15499)

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 for SH2A/WindRiver. 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

WindRiver C compiler can generate appropriate interrupt handling code for a C function decorated with the `#pragma fast_interrupt` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

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

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

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

Vector Number	Label
4–511	`_os_wrapper_VVVV`
eg :	`_os_wrapper_02F0` for vector number 90

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).

The vector table entries beyond the last bound vector are conditioned such that if OS_TRIM_UNUSED_VECTORS is defined they will not be present in the vector table

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()`. The `#pragma interrupt` designates function as an interrupt function. If register banking is to be used for all priorities then the `#pragma fast_interrupt` function qualifier should be used (please note this is not mentioned in the Wind River compiler documentation).

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. on 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 executed instructions is dependant upon the configuration of the memory and the code location. 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.src` 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.

3.1.10 Flash Security ID

To protect the contents of internal ROM some SH2A variants such as the SH7254x support an 8 byte security number located at memory address 0x60.

This address falls within the interrupt vector table range when it is located at its default setting (i.e. the VBR is zero). The user can enter an 8 byte security number at this address by defining the three symbols OS_SECURITY_ID (to enable the security ID setting), OS_SECURITY_ID_VAL_0x60 (to give the 4 bytes at 0x60-0x63), and OS_SECURITY_ID_VAL_0x64 (to give the 4 bytes at 0x64-0x67).

For example, assembling `osgen.s` with the command line options:

```
-DOS_SECURITY_ID
-DOS_SECURITY_ID_VAL_0x60=0x1234aabb
-DOS_SECURITY_ID_VAL_0x64=0xccdd8765
```

will insert the security number with bytes 0x60-0x63 having values 0x12,0x34,0xaa,0xbb, and bytes 0x64-0x67 having values 0xcc, 0xdd,0x87,0x65.

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().

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): 72

Timing

API max usage (bytes): 72

Extended

API max usage (bytes): 96

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

3.4 Floating point

SH2A-FPU CPUs contain a 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>*`\sh2awr\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 and recompiled selecting the `%CPU_TYPE%` as SH2AEH. 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 by selecting the `%CPU_TYPE%` as SH2AES. Double precession floating point is supported by selecting %CPU_TYPE% as SH2AEH and compilation option `-DOS_FPD`.

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 (16 and 32-bit) hardware counters. RTA-OSEK assumes that the stopwatch counter uses a 32-bit counter. If this is not the case then the remaining bits should be maintained in software as demonstrated in the example application.

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 SH72546**
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	46	46	46	46	46	46
	ROM	185	177	188	226	226	237
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	66	66	66	66	66	66
	ROM	257	249	265	298	298	314
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	84	84	84	84	84	84
	ROM	299	299	315	351	351	367
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

4.2.2 ROM and RAM for OSEK OS Objects

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

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

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	36	36	36	36	36	36
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	40	40	40	40	40	40
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	48	56	n/a	48	56
ECC1, Integer task	RAM	n/a	n/a	n/a	60	60	60
	ROM	n/a	n/a	n/a	60	60	60
ECC1, floating-point task	RAM	n/a	n/a	n/a	132	132	132
	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	134
	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	72	72	72	72	72	72
	ROM	63	63	63	63	63	63
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	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	77	77	77	77	77	77
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	20	20	20	20	20	20

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	64	64	64	64	64	64
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	52	52	52	52	52	52
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	60	68	n/a	60	68
ECC1, Integer task	RAM	n/a	n/a	n/a	72	72	72
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating-point task	RAM	n/a	n/a	n/a	144	144	144
	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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	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	146
	ROM	n/a	n/a	n/a	n/a	n/a	80
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	86	86	86	86	86	86
Category 2 ISR, floating-point	RAM	84	84	84	84	84	84
	ROM	98	98	98	98	98	98
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	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	77	77	77	77	77	77
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	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	64	64	64	64	64	64
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16

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

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	16	16	16	16	16	16
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	68	76	n/a	68	76
ECC1, Integer task	RAM	n/a	n/a	n/a	76	76	76
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	148	148	148
	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	150
	ROM	n/a	n/a	n/a	n/a	n/a	88
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	98	98	98	98	98	98
Category 2 ISR, floating-point	RAM	88	88	88	88	88	88
	ROM	110	110	110	110	110	110
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Alarm	RAM	12	12	12	12	12	12
	ROM	34	34	34	34	34	34
Counter	RAM	4	4	4	4	4	4
	ROM	81	81	81	81	81	81
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	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	64	64	64	64	64	64
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Arrivalpoint (writable)	RAM	20	20	20	20	20	20
	ROM	20	20	20	20	20	20
Schedule	RAM	20	20	20	20	20	20
	ROM	44	44	44	44	44	44
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

4.2.3 Size of Linkable Modules

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

In some cases there are multiple variants of particular API calls. This is because the offline configuration of RTA-OSEK can determine when

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

The call variants are as follows:

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

Variant	Description
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	132	183	224	139	193	250
	NS		106	163	201	116	169	225
	KL	2	77	124	163	80	133	185
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	19	19	19	19	19	19
ChainTask	SWL	1, 8	123	178	215	132	185	234
	SWH	1, 9	144	194	231	154	200	255
	NSL	8	123	178	215	132	185	234
	NSH	9	134	186	223	144	194	251
Schedule			86	86	103	86	86	103
GetTaskID			25	25	25	25	25	25
GetTaskState			78	78	78	91	91	91
EnableAllInterrupts			26	26	26	26	26	26
DisableAllInterrupts			25	25	25	25	25	25
ResumeAllInterrupts			44	44	44	44	44	44
SuspendAllInterrupts			43	43	43	43	43	43
ResumeOSInterrupts			38	38	38	38	38	38
SuspendOSInterrupts			55	55	55	55	55	55
GetResource	Task	7	25	25	29	25	25	29
	Combined	6	66	66	66	66	66	66
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	70	70	70	70	70	70
	Combined	6	157	157	157	157	157	157
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	114	114	188
	NS		n/a	n/a	n/a	90	90	164
	NS1i	10	n/a	n/a	n/a	54	n/a	n/a
	KL	2	n/a	n/a	n/a	54	54	132
	KL1i	2, 10	n/a	n/a	n/a	21	n/a	n/a
ClearEvent			n/a	n/a	n/a	54	54	54

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetEvent			n/a	n/a	n/a	10	10	10
WaitEvent	<default>		n/a	n/a	n/a	203	203	371
	fp	11	n/a	n/a	n/a	234	234	420
	1i	10	n/a	n/a	n/a	17	n/a	n/a
GetAlarmBase			53	53	53	53	53	53
GetAlarm			88	88	88	88	88	88
SetRelAlarm			407	407	407	407	407	407
SetAbsAlarm			434	434	434	434	434	434
CancelAlarm			80	80	80	80	80	80
InitCounter			55	55	55	55	55	55
GetCounterValue			67	67	67	67	67	67
GetScheduleTableStatus		34	54	82	82	54	82	82
NextScheduleTable		34	79	178	178	79	178	178
StartScheduleTable		34	117	178	178	117	178	178
StopScheduleTable		34	86	118	118	86	118	118
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		10	10	10	11	11	11
ScheduleTable expiry point	Tick counter		10	10	10	10	10	10
ScheduleTable expiry point	Final		32	32	32	32	32	32
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	38	38	38	38	38	38
Process container	Non-Yielding	33	28	28	28	28	28	28
osek_tick_alarm	<default>		73	73	73	73	73	73
	KL	2	34	34	34	34	34	34
osek_incr_counter			24	24	24	24	24	24
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			134	134	134	134	134	134
ShutdownOS	NoHook	12	34	34	34	34	34	34
	Hook	13	42	42	42	42	42	42
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			31	31	31	31	31	31
StopCOM			19	19	19	19	19	19
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	69	69	69	174	174	174

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	174	174	174	174	174	174
GetMessageResource			36	36	36	36	36	36
ReleaseMessageResource			39	39	39	39	39	39
GetMessageStatus			42	42	42	42	42	42
SendMessage	SW CCCA	1, 14	97	97	97	221	221	221
	SW CCCB	1, 15	203	203	203	221	221	221
	NS CCCA	14	97	97	97	221	221	221
	NS CCCB	15	203	203	203	221	221	221
	KL CCCA	2, 14	66	66	66	188	188	188
	KL CCCB	2, 15	175	175	175	188	188	188
main_dispatch	NoHook	12	114	114	149	114	114	149
	Hook	13	151	151	181	151	151	181
sub_dispatch	B1LF	19	28	28	28	28	28	28
	B1HI	20	97	97	97	97	97	97
	B1HF	21	108	108	108	108	108	108
	B2LI	22	n/a	71	101	n/a	71	101
	B2LF	23	n/a	87	113	n/a	87	113
	B2HI	24	n/a	224	289	n/a	224	289
	B2HF	25	n/a	235	300	n/a	235	300
	E1HI	26	n/a	n/a	n/a	322	322	389
	E1HF	27	n/a	n/a	n/a	332	332	395
	E2HI	28	n/a	n/a	n/a	n/a	n/a	389
	E2HF	29	n/a	n/a	n/a	n/a	n/a	395
ErrorHook support		16	40	40	40	40	40	40
	ServiceID	17	49	49	49	49	49	49
	Parameters	18	60	60	60	60	60	60
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	131	231	276	149	249	310
	NS		110	207	252	126	228	285
	KL	2	73	171	209	90	198	253
ChainTaskset	SWL	1, 8	127	223	268	136	241	296
	SWH	1, 9	158	260	303	166	276	326
	NSL	8	127	223	268	136	241	296
	NSH	9	151	254	296	158	270	323
GetTasksetRef			8	8	8	8	8	8

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
MergeTaskset			49	49	49	49	49	49
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			51	51	51	51	51	51
TestSubTaskset			55	55	55	55	55	55
TestEquivalentTaskset			53	53	53	53	53	53
TickSchedule	SW	1	184	157	157	157	157	157
	NS		156	130	130	130	130	130
	KL	2	127	99	99	99	99	99
AdvanceSchedule	SW	1	176	147	147	147	147	147
	NS		148	118	118	118	118	118
	KL	2	119	85	85	85	85	85
StartSchedule			73	73	73	73	73	73
StopSchedule			57	57	57	57	57	57
GetScheduleStatus			85	85	85	85	85	85
GetScheduleValue			63	63	63	63	63	63
GetScheduleNext			8	8	8	8	8	8
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			6	6	6	6	6	6
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			8	8	8	8	8	8
SetArrivalpointNext			6	6	6	6	6	6
TestArrivalpointWritable			29	29	29	29	29	29
GetExecutionTime			4	4	4	4	4	4
GetLargestExecutionTime			8	8	8	8	8	8
ResetLargestExecutionTime			4	4	4	4	4	4
GetStackOffset			15	15	15	15	15	15

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	132	183	224	139	193	250
	NS		106	163	201	116	169	225
	KL	2	77	124	163	80	133	185
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	19	19	19	19	19	19
ChainTask	SWL	1, 8	123	178	215	132	185	234
	SWH	1, 9	144	194	231	154	200	255
	NSL	8	123	178	215	132	185	234
	NSH	9	134	186	223	144	194	251
Schedule			102	102	118	102	102	118
GetTaskID			25	25	25	25	25	25
GetTaskState			78	78	78	91	91	91
EnableAllInterrupts			26	26	26	26	26	26
DisableAllInterrupts			25	25	25	25	25	25
ResumeAllInterrupts			44	44	44	44	44	44
SuspendAllInterrupts			43	43	43	43	43	43
ResumeOSInterrupts			38	38	38	38	38	38
SuspendOSInterrupts			55	55	55	55	55	55
GetResource	Task	7	25	25	29	25	25	29
	Combined	6	66	66	66	66	66	66
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	87	87	87	87	87	87
	Combined	6	193	193	193	193	193	193
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	114	114	188
	NS		n/a	n/a	n/a	90	90	164
	NS1i	10	n/a	n/a	n/a	54	n/a	n/a
	KL	2	n/a	n/a	n/a	54	54	132
	KL1i	2, 10	n/a	n/a	n/a	21	n/a	n/a
ClearEvent			n/a	n/a	n/a	54	54	54
GetEvent			n/a	n/a	n/a	10	10	10
WaitEvent	<default>		n/a	n/a	n/a	250	250	405
	fp	11	n/a	n/a	n/a	280	280	452

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	87	n/a	n/a
GetAlarmBase			53	53	53	53	53	53
GetAlarm			88	88	88	88	88	88
SetRelAlarm			407	407	407	407	407	407
SetAbsAlarm			434	434	434	434	434	434
CancelAlarm			80	80	80	80	80	80
InitCounter			55	55	55	55	55	55
GetCounterValue			67	67	67	67	67	67
GetScheduleTableStatus		34	54	82	82	54	82	82
NextScheduleTable		34	79	178	178	79	178	178
StartScheduleTable		34	117	178	178	117	178	178
StopScheduleTable		34	86	118	118	86	118	118
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		10	10	10	11	11	11
ScheduleTable expiry point	Tick counter		10	10	10	10	10	10
ScheduleTable expiry point	Final		32	32	32	32	32	32
GetISRID		4	41	41	41	41	41	41
Process container	Yielding	32	38	38	38	38	38	38
Process container	Non-Yielding	33	28	28	28	28	28	28
osek_tick_alarm	<default>		73	73	73	73	73	73
	KL	2	34	34	34	34	34	34
osek_incr_counter			24	24	24	24	24	24
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			164	164	164	164	164	164
ShutdownOS	NoHook	12	34	34	34	34	34	34
	Hook	13	42	42	42	42	42	42
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			31	31	31	31	31	31
StopCOM			19	19	19	19	19	19
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	69	69	69	174	174	174
	CCCB	15	174	174	174	174	174	174
GetMessageResource			36	36	36	36	36	36
ReleaseMessageResource			39	39	39	39	39	39

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			42	42	42	42	42	42
SendMessage	SW CCCA	1, 14	97	97	97	221	221	221
	SW CCCB	1, 15	203	203	203	221	221	221
	NS CCCA	14	97	97	97	221	221	221
	NS CCCB	15	203	203	203	221	221	221
	KL CCCA	2, 14	66	66	66	188	188	188
	KL CCCB	2, 15	175	175	175	188	188	188
main_dispatch	NoHook	12	132	132	164	132	132	164
	Hook	13	178	178	206	178	178	206
sub_dispatch	B1LF	19	23	23	23	23	23	23
	B1HI	20	98	98	98	98	98	98
	B1HF	21	111	111	111	111	111	111
	B2LI	22	n/a	49	75	n/a	49	75
	B2LF	23	n/a	63	86	n/a	63	86
	B2HI	24	n/a	203	275	n/a	203	275
	B2HF	25	n/a	216	283	n/a	216	283
	E1HI	26	n/a	n/a	n/a	330	330	399
	E1HF	27	n/a	n/a	n/a	338	338	407
	E2HI	28	n/a	n/a	n/a	n/a	n/a	399
	E2HF	29	n/a	n/a	n/a	n/a	n/a	407
ErrorHook support		16	40	40	40	40	40	40
	ServiceID	17	49	49	49	49	49	49
	Parameters	18	60	60	60	60	60	60
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	70	70	70	70	70	70
Timing_termination		4	76	76	76	76	76	76
ActivateTaskset	SW	1	131	231	276	149	249	310
	NS		110	207	252	126	228	285
	KL	2	73	171	209	90	198	253
ChainTaskset	SWL	1, 8	127	223	268	136	241	296
	SWH	1, 9	158	260	303	166	276	326
	NSL	8	127	223	268	136	241	296
	NSH	9	151	254	296	158	270	323
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			49	49	49	49	49	49
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			51	51	51	51	51	51

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			55	55	55	55	55	55
TestEquivalentTaskset			53	53	53	53	53	53
TickSchedule	SW	1	184	157	157	157	157	157
	NS		156	130	130	130	130	130
	KL	2	127	99	99	99	99	99
AdvanceSchedule	SW	1	176	147	147	147	147	147
	NS		148	118	118	118	118	118
	KL	2	119	85	85	85	85	85
StartSchedule			73	73	73	73	73	73
StopSchedule			57	57	57	57	57	57
GetScheduleStatus			85	85	85	85	85	85
GetScheduleValue			63	63	63	63	63	63
GetScheduleNext			8	8	8	8	8	8
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			6	6	6	6	6	6
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			8	8	8	8	8	8
SetArrivalpointNext			6	6	6	6	6	6
TestArrivalpointWritable			29	29	29	29	29	29
GetExecutionTime			90	90	90	90	90	90
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			12	12	12	12	12	12
GetStackOffset			15	15	15	15	15	15

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	208	261	304	213	262	318
	NS		250	302	341	256	310	359
	KL	2	143	195	234	149	197	252
TerminateTask	LExt	3	94	94	94	94	94	94

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	120	120	120	120	120	120
ChainTask	SWL	1, 8	240	296	333	244	298	355
	SWH	1, 9	267	315	352	271	321	370
	NSL	8	293	354	393	298	356	411
	NSH	9	317	367	404	319	375	426
Schedule			204	204	220	204	204	220
GetTaskID			38	38	38	38	38	38
GetTaskState			195	195	195	195	195	195
EnableAllInterrupts			40	40	40	40	40	40
DisableAllInterrupts			39	39	39	39	39	39
ResumeAllInterrupts			87	87	87	87	87	87
SuspendAllInterrupts			60	60	60	60	60	60
ResumeOSInterrupts			85	85	85	85	85	85
SuspendOSInterrupts			72	72	72	72	72	72
GetResource	Task	7	298	298	254	298	298	254
	Combined	6	282	282	282	282	282	282
	CLEx	3	252	252	252	252	252	252
ReleaseResource	Task	7	285	285	285	285	285	285
	Combined	6	364	364	364	364	364	364
	CLEx	3	243	243	243	243	243	243
SetEvent	SW	1	n/a	n/a	n/a	251	251	319
	NS		n/a	n/a	n/a	287	287	361
	NS1i	10	n/a	n/a	n/a	188	n/a	n/a
	KL	2	n/a	n/a	n/a	191	191	260
	KL1i	2, 10	n/a	n/a	n/a	142	n/a	n/a
ClearEvent			n/a	n/a	n/a	118	118	118
GetEvent			n/a	n/a	n/a	135	135	135
WaitEvent	<default>		n/a	n/a	n/a	333	333	489
	fp	11	n/a	n/a	n/a	361	361	528
	1i	10	n/a	n/a	n/a	180	n/a	n/a
GetAlarmBase			149	149	149	149	149	149
GetAlarm			151	151	151	151	151	151
SetRelAlarm			519	519	519	519	519	519
SetAbsAlarm			531	531	531	531	531	531
CancelAlarm			141	141	141	141	141	141
InitCounter			184	184	184	184	184	184
GetCounterValue			161	161	161	161	161	161

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	72	92	92	72	92	92
NextScheduleTable		34	92	189	189	92	189	189
StartScheduleTable		34	128	188	188	128	188	188
StopScheduleTable		34	97	130	130	97	130	130
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		10	10	10	11	11	11
ScheduleTable expiry point	Tick counter		10	10	10	10	10	10
ScheduleTable expiry point	Final		32	32	32	32	32	32
GetISRID		4	55	55	55	55	55	55
Process container	Yielding	32	38	38	38	38	38	38
Process container	Non-Yielding	33	28	28	28	28	28	28
osek_tick_alarm	<default>		103	103	103	103	103	103
	KL	2	34	34	34	34	34	34
osek_incr_counter			24	24	24	24	24	24
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			176	176	176	176	176	176
ShutdownOS	NoHook	12	43	43	43	43	43	43
	Hook	13	48	48	48	48	48	48
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			42	42	42	42	42	42
StopCOM			42	42	42	42	42	42
ReadFlag			28	28	28	28	28	28
ResetFlag			32	32	32	32	32	32
ReceiveMessage	CCCA	14	144	144	144	250	250	250
	CCCB	15	250	250	250	250	250	250
GetMessageResource			84	84	84	84	84	84
ReleaseMessageResource			88	88	88	88	88	88
GetMessageStatus			81	81	81	81	81	81
SendMessage	SW CCCA	1, 14	180	180	180	303	303	303
	SW CCCB	1, 15	287	287	287	303	303	303
	NS CCCA	14	180	180	180	303	303	303
	NS CCCB	15	287	287	287	303	303	303
	KL CCCA	2, 14	136	136	136	258	258	258
	KL CCCB	2, 15	243	243	243	258	258	258
main_dispatch	NoHook	12	132	132	164	132	132	164

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	178	178	206	178	178	206
sub_dispatch	B1LF	19	23	23	23	23	23	23
	B1HI	20	102	102	102	102	102	102
	B1HF	21	111	111	111	111	111	111
	B2LI	22	n/a	49	75	n/a	49	75
	B2LF	23	n/a	63	86	n/a	63	86
	B2HI	24	n/a	203	275	n/a	203	275
	B2HF	25	n/a	216	283	n/a	216	283
	E1HI	26	n/a	n/a	n/a	330	330	399
	E1HF	27	n/a	n/a	n/a	338	338	407
	E2HI	28	n/a	n/a	n/a	n/a	n/a	399
	E2HF	29	n/a	n/a	n/a	n/a	n/a	407
ErrorHook support		16	100	100	100	100	100	100
	ServiceID	17	109	109	109	109	109	109
	Parameters	18	124	124	124	124	124	124
validity_checks		3	30	30	30	30	30	30
Timing_dispatch		4	70	70	70	70	70	70
Timing_termination		4	76	76	76	76	76	76
ActivateTaskset	SW	1	275	328	370	287	341	404
	NS		312	365	413	324	382	448
	KL	2	195	232	282	213	263	324
ChainTaskset	SWL	1, 8	322	356	407	324	368	425
	SWH	1, 9	355	417	450	357	421	480
	NSL	8	372	410	457	378	430	493
	NSH	9	407	477	510	411	477	603
GetTasksetRef			104	104	104	104	104	104
MergeTaskset			199	199	199	199	199	199
AssignTaskset			122	122	122	122	122	122
RemoveTaskset			200	200	200	200	200	200
TestSubTaskset			199	199	199	199	199	199
TestEquivalentTaskset			197	197	197	197	197	197
TickSchedule	SW	1	269	250	250	250	250	250
	NS		310	302	302	302	302	302
	KL	2	205	182	182	182	182	182
AdvanceSchedule	SW	1	273	256	256	256	256	256
	NS		314	305	305	305	305	305
	KL	2	208	186	186	186	186	186

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			194	194	194	194	194	194
StopSchedule			148	148	148	148	148	148
GetScheduleStatus			180	180	180	180	180	180
GetScheduleValue			154	154	154	154	154	154
GetScheduleNext			68	68	68	68	68	68
SetScheduleNext			128	128	128	128	128	128
GetArrivalpointDelay			97	97	97	97	97	97
SetArrivalpointDelay			109	109	109	109	109	109
GetArrivalpointTasksetRef			91	91	91	91	91	91
GetArrivalpointNext			97	97	97	97	97	97
SetArrivalpointNext			135	135	135	135	135	135
TestArrivalpointWritable			107	107	107	107	107	107
GetExecutionTime			135	135	135	135	135	135
GetLargestExecutionTime			86	86	86	86	86	86
ResetLargestExecutionTime			83	83	83	83	83	83
GetStackOffset			15	15	15	15	15	15

Notes

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

Number	Note
16	Linked only if `USEGETSERVICEID = FALSE` and `USEPARAMETERACCESS = FALSE`
17	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = FALSE`
18	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = TRUE`
19	Linked only for basic, single-activation, lightweight, floating-point tasks
20	Linked only for basic, single-activation, heavyweight, integer tasks
21	Linked only for basic, single-activation, heavyweight, floating-point tasks
22	Linked only for basic, multiple-activation, lightweight, integer tasks
23	Linked only for basic, multiple-activation, lightweight, floating-point tasks
24	Linked only for basic, multiple-activation, heavyweight, integer tasks
25	Linked only for basic, multiple-activation, heavyweight, floating-point tasks
26	Linked only for extended, unique priority, integer tasks
27	Linked only for extended, unique priority, floating-point tasks
28	Linked only for extended, shared priority, integer tasks
29	Linked only for extended, shared priority, floating-point tasks
30	Implemented as a macro, so no code is linked
31	Not required on some targets
32	Container for 2 process functions, not highest priority
33	Container for 2 process functions, highest or APPMODE or ISR
34	code varies with number of schedule tables; example uses 2 schedule tables

4.2.4 Reserved Hardware Resources

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

4.3 Performance

4.3.1 Execution Times for RTA-OSEK API Calls

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	135	169	215	139	169	215
	NS	117	153	197	119	149	197
	KL	73	107	155	73	111	159
TerminateTask	LExt	0	0	0	0	0	0
	H	177	175	177	175	177	177
ChainTask	SWL	329	385	465	373	405	475
	SWH	385	435	521	429	457	527
	NSL	329	385	465	371	405	477
	NSH	377	431	515	423	449	527
Schedule	SW	103	105	117	105	105	119
GetTaskID		51	51	51	45	47	45
GetTaskState		113	111	111	125	125	125
EnableAllInterrupts		45	45	45	45	45	45
DisableAllInterrupts		59	59	59	61	59	59
ResumeAllInterrupts		59	59	59	59	59	59
SuspendAllInterrupts		71	71	71	71	71	73
ResumeOSInterrupts		57	57	57	57	57	57
SuspendOSInterrupts		71	71	71	71	71	73
GetResource	Task	57	57	59	59	57	57
	Combined	89	89	87	87	85	83
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	99	99	97	97	99	97
	Combined	131	131	131	133	131	133
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	133	131	137
	NS	n/a	n/a	n/a	117	119	119
	KL	n/a	n/a	n/a	69	67	81
ClearEvent		n/a	n/a	n/a	87	91	91
GetEvent		n/a	n/a	n/a	33	33	33
WaitEvent	<default>	n/a	n/a	n/a	499	501	531
	fp	n/a	n/a	n/a	499	499	539
GetAlarmBase		115	113	111	117	115	115
GetAlarm		121	123	121	127	123	121

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		163	161	167	159	161	159
SetAbsAlarm		159	159	163	159	161	157
CancelAlarm		103	99	99	101	101	105
InitCounter		99	99	99	99	101	99
GetCounterValue		99	101	103	103	99	97
osek_tick_alarm	<default>	115	115	115	115	115	115
	KL	53	53	53	55	55	55
osek_incr_counter		37	41	37	35	35	35
GetActiveApplicationMode		27	27	27	27	27	27
StartOS		1085	1083	1083	1081	1083	1081
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	49	49	47	45	45	45
InitCOM		31	27	25	29	27	29
CloseCOM		29	29	25	27	29	27
StartCOM		69	69	67	145	147	143
StopCOM		39	39	37	37	39	37
ReadFlag		n/a	n/a	n/a	31	31	31
ResetFlag		n/a	n/a	n/a	27	27	27
ReceiveMessage		85	85	85	283	281	289
GetMessageResource		n/a	n/a	n/a	145	141	139
ReleaseMessageResource		n/a	n/a	n/a	155	157	153
GetMessageStatus		n/a	n/a	n/a	65	67	61
SendMessage	SW	215	253	299	413	449	493
	NS	193	233	277	393	427	473
	KL	127	161	209	325	359	409
ActivateTaskset	SW	121	467	519	125	461	529
	NS	99	445	459	101	441	525
	KL	61	411	499	71	413	451
	SW2	121	463	513	125	461	529
	NS2	99	445	459	101	441	525
	KL2	67	411	499	71	415	451
ChainTaskset	SWL	311	675	723	355	687	793
	SWH	381	739	859	417	749	813
	NSL	311	673	725	351	685	795
	NSH	377	731	849	419	747	801
GetTasksetRef		41	43	43	41	41	41

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		91	91	91	91	91	91
AssignTaskset		33	33	33	33	33	33
RemoveTaskset		91	89	89	89	87	87
TestSubTaskset		101	103	101	101	101	101
TestEquivalentTaskset		95	95	95	95	95	95
TickSchedule	SW	175	581	667	243	599	625
	NS	155	557	645	215	575	603
	KL	121	517	603	175	535	561
	SW2	175	579	667	239	581	619
	NS2	153	555	643	213	555	595
	KL2	121	515	603	175	517	555
AdvanceSchedule	SW	177	573	657	235	587	621
	NS	145	537	627	197	557	585
	KL	115	509	599	167	529	557
	SW2	173	569	653	229	567	607
	NS2	145	539	627	197	543	577
	KL2	115	511	599	167	513	549
StartSchedule		113	111	111	111	107	111
StopSchedule		95	95	97	93	93	97
GetScheduleStatus		115	115	113	113	115	113
GetScheduleValue		105	103	101	101	107	103
GetScheduleNext		33	33	35	33	33	33
SetScheduleNext		33	33	33	33	31	33
GetArrivalpointDelay		31	31	31	31	31	31
SetArrivalpointDelay		29	29	29	29	29	29
GetArrivalpointTasksetRef		29	29	29	31	29	31
GetArrivalpointNext		29	29	29	29	29	33
SetArrivalpointNext		29	29	29	29	29	29
TestArrivalpointWritable		39	39	39	39	37	39
GetExecutionTime		31	27	27	35	27	29
GetLargestExecutionTime		37	37	37	37	37	37
ResetLargestExecutionTime		27	27	27	25	27	25
GetStackOffset		35	35	35	35	35	35

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	137	177	213	139	159	211
	NS	121	161	193	121	141	191
	KL	73	115	153	75	105	151
TerminateTask	LExt	0	0	0	0	0	0
	H	441	437	439	443	435	443
ChainTask	SWL	629	699	757	695	715	787
	SWH	681	741	807	743	753	831
	NSL	631	699	757	691	715	783
	NSH	673	735	797	741	751	831
Schedule	SW	111	107	123	105	107	121
GetTaskID		53	51	53	47	45	45
GetTaskState		111	111	115	125	125	127
EnableAllInterrupts		45	45	45	45	45	45
DisableAllInterrupts		59	59	61	59	61	55
ResumeAllInterrupts		59	59	59	59	59	59
SuspendAllInterrupts		71	73	71	73	71	71
ResumeOSInterrupts		57	57	57	57	57	57
SuspendOSInterrupts		71	73	71	73	71	71
GetResource	Task	57	57	59	57	59	59
	Combined	85	87	87	83	89	89
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	99	97	99	97	99	99
	Combined	129	129	129	129	131	133
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	133	137	139
	NS	n/a	n/a	n/a	119	115	119
	KL	n/a	n/a	n/a	67	65	81
ClearEvent		n/a	n/a	n/a	91	87	91
GetEvent		n/a	n/a	n/a	33	33	33
WaitEvent	<default>	n/a	n/a	n/a	777	765	807
	fp	n/a	n/a	n/a	783	773	815
GetAlarmBase		111	113	115	111	109	109
GetAlarm		125	125	121	123	121	123

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		161	161	163	159	159	161
SetAbsAlarm		161	161	159	155	155	159
CancelAlarm		101	99	103	103	103	101
InitCounter		99	99	99	99	101	99
GetCounterValue		99	99	99	101	99	99
osek_tick_alarm	<default>	115	115	115	115	115	115
	KL	53	53	53	57	55	55
osek_incr_counter		37	37	37	35	35	35
GetActiveApplicationMode		21	21	21	23	23	23
StartOS		2977	2985	2979	2985	2981	2979
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	49	49	49	47	45	45
InitCOM		27	29	25	29	25	25
CloseCOM		25	25	25	27	29	27
StartCOM		69	71	71	149	145	143
StopCOM		37	37	37	37	39	37
ReadFlag		n/a	n/a	n/a	31	31	31
ResetFlag		n/a	n/a	n/a	27	27	27
ReceiveMessage		85	85	85	281	279	283
GetMessageResource		n/a	n/a	n/a	141	141	141
ReleaseMessageResource		n/a	n/a	n/a	155	155	153
GetMessageStatus		n/a	n/a	n/a	61	61	67
SendMessage	SW	217	257	297	413	437	489
	NS	195	237	271	391	417	467
	KL	129	167	207	329	353	399
ActivateTaskset	SW	119	463	519	123	461	523
	NS	95	445	457	101	439	533
	KL	55	411	499	67	415	449
	SW2	119	463	519	123	461	523
	NS2	95	445	457	101	439	533
	KL2	55	411	499	67	415	449
ChainTaskset	SWL	611	983	1015	679	995	1109
	SWH	681	1039	1145	733	1047	1111
	NSL	613	979	1015	673	995	1113
	NSH	671	1033	1133	727	1043	1109
GetTasksetRef		43	43	41	41	43	41

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		91	91	91	91	91	91
AssignTaskset		33	33	33	33	33	33
RemoveTaskset		89	89	89	87	89	89
TestSubTaskset		101	103	103	101	101	101
TestEquivalentTaskset		95	95	95	95	95	95
TickSchedule	SW	175	579	665	237	601	623
	NS	155	557	643	215	579	601
	KL	121	515	605	171	535	563
	SW2	175	579	665	237	585	615
	NS2	153	555	641	213	561	591
	KL2	121	515	605	171	521	555
AdvanceSchedule	SW	171	567	657	225	585	609
	NS	143	537	631	201	553	585
	KL	111	509	599	175	527	555
	SW2	171	567	655	223	571	601
	NS2	143	537	625	199	539	577
	KL2	111	509	597	171	513	547
StartSchedule		111	113	111	109	109	111
StopSchedule		95	95	97	95	95	93
GetScheduleStatus		113	115	113	115	113	113
GetScheduleValue		101	105	101	105	99	99
GetScheduleNext		33	33	33	33	35	33
SetScheduleNext		33	33	33	31	31	31
GetArrivalpointDelay		31	31	31	33	31	31
SetArrivalpointDelay		29	27	29	29	29	29
GetArrivalpointTasksetRef		29	29	29	29	31	29
GetArrivalpointNext		29	29	29	29	29	29
SetArrivalpointNext		29	29	29	27	29	29
TestArrivalpointWritable		39	39	39	39	39	37
GetExecutionTime		169	163	169	165	163	163
GetLargestExecutionTime		47	47	47	47	47	47
ResetLargestExecutionTime		39	37	39	37	35	35
GetStackOffset		35	35	37	37	35	35

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	343	377	413	347	371	415
	NS	377	415	445	383	409	451
	KL	297	331	367	307	327	369
TerminateTask	LExt	507	501	503	497	503	501
	H	539	543	541	537	541	541
ChainTask	SWL	927	989	1057	979	1003	1087
	SWH	969	1021	1087	1025	1045	1113
	NSL	967	1023	1091	1019	1045	1125
	NSH	1009	1063	1133	1055	1079	1149
Schedule	SW	167	171	183	167	167	183
GetTaskID		61	61	61	61	61	61
GetTaskState		353	353	357	361	361	361
EnableAllInterrupts		57	57	57	57	57	57
DisableAllInterrupts		67	69	73	67	71	71
ResumeAllInterrupts		71	79	73	71	73	73
SuspendAllInterrupts		85	83	83	85	83	83
ResumeOSInterrupts		71	73	73	71	73	73
SuspendOSInterrupts		85	83	83	85	83	83
GetResource	Task	513	509	285	557	551	323
	Combined	265	269	271	307	307	309
	CLEx	285	287	283	333	329	327
ReleaseResource	Task	277	271	271	323	317	313
	Combined	281	281	283	327	319	321
	CLEx	261	257	255	301	297	297
SetEvent	SW	n/a	n/a	n/a	381	381	379
	NS	n/a	n/a	n/a	399	395	393
	KL	n/a	n/a	n/a	341	345	341
ClearEvent		n/a	n/a	n/a	127	123	123
GetEvent		n/a	n/a	n/a	311	309	309
WaitEvent	<default>	n/a	n/a	n/a	921	923	957
	fp	n/a	n/a	n/a	929	929	961
GetAlarmBase		251	257	265	253	257	265
GetAlarm		261	261	275	259	259	273

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		317	321	327	315	321	333
SetAbsAlarm		297	295	309	295	297	313
CancelAlarm		243	239	257	241	239	259
InitCounter		347	345	363	345	347	367
GetCounterValue		231	227	229	227	229	231
osek_tick_alarm	<default>	137	143	139	143	143	143
	KL	53	55	53	53	55	53
osek_incr_counter		35	35	35	37	37	37
GetActiveApplicationMode		21	21	21	23	23	23
StartOS		3101	3109	3109	3105	3105	3103
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	53	51	51	61	55	55
InitCOM		27	31	31	29	33	33
CloseCOM		31	27	27	29	29	29
StartCOM		85	87	85	163	163	159
StopCOM		57	55	55	59	59	59
ReadFlag		n/a	n/a	n/a	59	61	61
ResetFlag		n/a	n/a	n/a	55	57	57
ReceiveMessage		199	195	195	393	393	393
GetMessageResource		n/a	n/a	n/a	445	449	445
ReleaseMessageResource		n/a	n/a	n/a	449	443	445
GetMessageStatus		n/a	n/a	n/a	161	163	163
SendMessage	SW	527	561	599	725	745	797
	NS	559	597	625	765	785	831
	KL	457	493	523	647	669	709
ActivateTaskset	SW	513	869	1113	523	871	937
	NS	539	897	951	545	891	937
	KL	457	813	933	465	811	915
	SW2	513	869	1113	523	871	937
	NS2	539	897	951	545	891	937
	KL2	457	813	933	465	811	915
ChainTaskset	SWL	1117	1485	1551	1167	1499	1597
	SWH	1167	1565	1671	1221	1577	1679
	NSL	1155	1537	1771	1203	1551	1807
	NSH	1201	1727	1825	1257	1761	1731
GetTasksetRef		283	287	285	285	285	285

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		157	161	161	165	167	167
AssignTaskset		97	99	97	93	89	89
RemoveTaskset		161	155	159	161	161	161
TestSubTaskset		171	165	167	173	173	173
TestEquivalentTaskset		169	163	167	171	169	169
TickSchedule	SW	239	1025	1149	681	1045	1147
	NS	265	1053	1177	707	1069	1169
	KL	181	963	1087	621	987	1085
	SW2	239	1025	1145	681	1023	1129
	NS2	265	1053	1173	707	1047	1151
	KL2	181	963	1083	621	963	1069
AdvanceSchedule	SW	233	1013	1141	671	1037	1135
	NS	257	1039	1161	693	1063	1157
	KL	183	963	1085	617	985	1081
	SW2	233	1013	1135	671	1013	1119
	NS2	257	1039	1157	693	1039	1141
	KL2	183	963	1081	617	963	1065
StartSchedule		179	181	181	181	181	181
StopSchedule		141	147	145	145	149	147
GetScheduleStatus		165	163	165	163	161	163
GetScheduleValue		151	153	153	159	155	157
GetScheduleNext		65	63	63	67	67	67
SetScheduleNext		107	107	107	113	109	109
GetArrivalpointDelay		83	85	83	83	81	81
SetArrivalpointDelay		101	101	97	99	95	95
GetArrivalpointTasksetRef		79	77	77	75	75	75
GetArrivalpointNext		75	77	77	73	73	73
SetArrivalpointNext		111	113	113	105	107	107
TestArrivalpointWritable		87	85	87	83	79	79
GetExecutionTime		201	195	189	187	193	187
GetLargestExecutionTime		267	265	265	261	263	263
ResetLargestExecutionTime		249	253	251	251	251	251
GetStackOffset		33	33	33	31	31	31

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	59	59	59	59	59	59
	Cat 2	83	311	311	307	307	311

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	59	59	59	59	59	59
	Cat 2	531	739	739	731	723	731

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	59	59	59	59	59	59
	Cat 2	523	731	723	739	735	735

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.

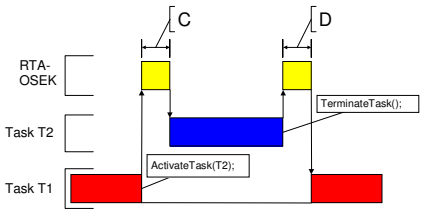

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

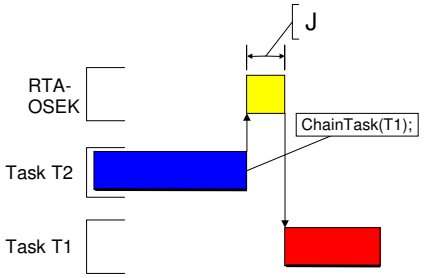

Figure 2: Task Chaining

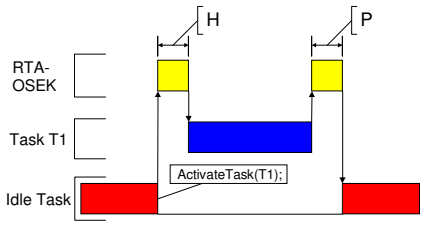

Figure 3: Task Activation from Idle Task

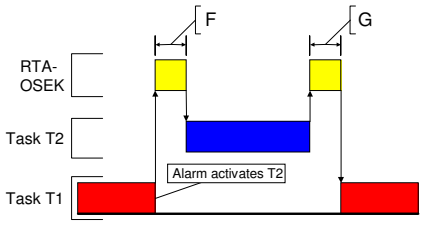

Figure 4: Task Activation from an Alarm

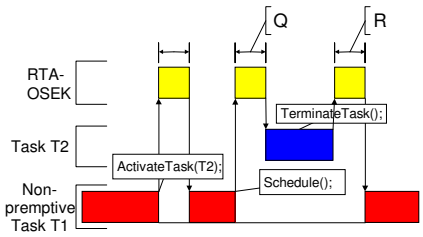

Figure 5: Non-Premptive Task Calls Schedule()

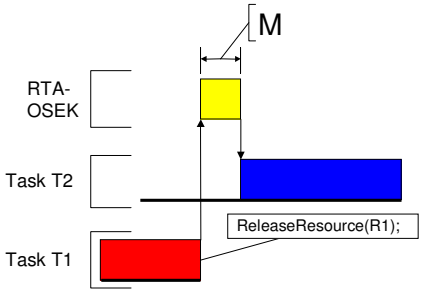

Figure 6: Blocked Task Activated by ReleaseResource()

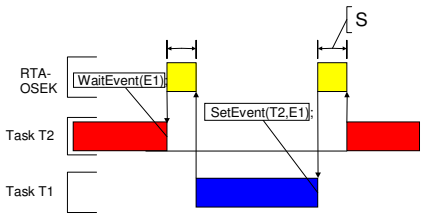

Figure 7: Waiting Task Activated by SetEvent()

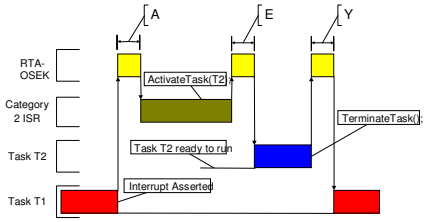

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	109	149	175	107	149	175
Figure 1: D	Heavy, Basic/Extended	177	217	241	221	221	243
ChainTask	Light, Basic	249	313	373	253	313	395
Figure 2: J	Heavy, Basic/Extended	543	637	725	593	645	747
Pre-emption	Light, Basic	243	303	393	251	295	399
Figure 1: C	Heavy, Basic/Extended	337	393	493	385	417	507
From idle task	Light, Basic	243	305	393	251	297	399
Figure 3: H	Heavy, Basic/Extended	337	395	495	387	419	507
Triggered by alarm	Light, Basic	353	415	503	361	405	507
Figure 4: F	Heavy, Basic/Extended	445	507	603	499	529	615
Schedule	Light, Basic	217	235	299	219	231	293
Figure 5: Q	Heavy, Basic/Extended	311	333	399	353	353	405
Release resource	Light, Basic	227	247	295	231	241	295
Figure 6: M	Heavy, Basic/Extended	321	345	397	365	363	407
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	593	595	701
From category 2 ISR	Light, Basic	307	413	463	393	407	457
Figure 8: E	Heavy, Basic/Extended	401	511	565	527	529	569

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	377	407	429	383	403	429
Figure 1: D	Heavy, Basic/Extended	445	467	491	481	469	497
ChainTask	Light, Basic	563	617	687	571	615	703
Figure 2: J	Heavy, Basic/Extended	1117	1183	1285	1159	1191	1301
Pre-emption	Light, Basic	433	471	555	431	481	565
Figure 1: C	Heavy, Basic/Extended	519	575	651	575	589	687
From idle task	Light, Basic	433	471	555	433	481	567
Figure 3: H	Heavy, Basic/Extended	519	577	651	577	591	687
Triggered by alarm	Light, Basic	543	583	665	541	589	677
Figure 4: F	Heavy, Basic/Extended	629	689	759	687	701	797
Schedule	Light, Basic	399	409	459	399	411	461
Figure 5: Q	Heavy, Basic/Extended	485	507	555	543	533	589
Release resource	Light, Basic	413	421	461	411	423	465
Figure 6: M	Heavy, Basic/Extended	499	519	557	555	545	593
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	777	765	869
From category 2 ISR	Light, Basic	745	843	881	835	839	887
Figure 8: E	Heavy, Basic/Extended	831	939	977	979	961	1013

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	493	523	545	489	527	549
Figure 1: D	Heavy, Basic/Extended	543	569	595	575	581	605
ChainTask	Light, Basic	851	905	975	855	903	991
Figure 2: J	Heavy, Basic/Extended	1499	1569	1661	1535	1579	1671
Pre-emption	Light, Basic	633	677	749	629	673	755
Figure 1: C	Heavy, Basic/Extended	725	775	847	771	797	881
From idle task	Light, Basic	629	675	747	629	671	753
Figure 3: H	Heavy, Basic/Extended	721	773	845	771	795	879
Triggered by alarm	Light, Basic	765	807	881	763	803	885
Figure 4: F	Heavy, Basic/Extended	853	901	977	899	929	1011
Schedule	Light, Basic	449	461	509	443	457	509
Figure 5: Q	Heavy, Basic/Extended	541	559	605	585	587	635
Release resource	Light, Basic	571	577	615	607	613	653
Figure 6: M	Heavy, Basic/Extended	663	675	711	749	743	779
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1075	1073	1179
From category 2 ISR	Light, Basic	787	881	917	859	875	921
Figure 8: E	Heavy, Basic/Extended	879	979	1013	1001	1005	1047

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	104	104	104	104	104	104
BCC1 lightweight, floating-point	108	108	108	108	108	108
BCC1 heavyweight, integer	112	112	112	112	112	112
BCC1 heavyweight, floating-point	176	176	176	176	176	176
BCC2 lightweight, integer	n/a	112	112	n/a	112	112
BCC2 lightweight, floating-point	n/a	112	112	n/a	112	112
BCC2 heavyweight, integer	n/a	192	192	n/a	192	192
BCC2 heavyweight, floating-point	n/a	192	192	n/a	192	192
ECC1 heavyweight, integer	n/a	n/a	n/a	208	208	208
ECC1 heavyweight, floating-point	n/a	n/a	n/a	208	208	208
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	208
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	208
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	104	104	104	104	104	104
BCC1 lightweight, floating-point	108	108	108	108	108	108
BCC1 heavyweight, integer	112	112	112	112	112	112
BCC1 heavyweight, floating-point	176	176	176	176	176	176
BCC2 lightweight, integer	n/a	112	112	n/a	112	112
BCC2 lightweight, floating-point	n/a	112	112	n/a	112	112
BCC2 heavyweight, integer	n/a	192	192	n/a	192	192
BCC2 heavyweight, floating-point	n/a	192	192	n/a	192	192
ECC1 heavyweight, integer	n/a	n/a	n/a	208	208	208
ECC1 heavyweight, floating-point	n/a	n/a	n/a	208	208	208
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	208
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	208

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	88	88	88	88	88	88
BCC1 lightweight, floating-point	92	92	92	92	92	92
BCC1 heavyweight, integer	160	160	160	160	160	160
BCC1 heavyweight, floating-point	160	160	160	160	160	160
BCC2 lightweight, integer	n/a	92	92	n/a	92	92
BCC2 lightweight, floating-point	n/a	92	92	n/a	92	92
BCC2 heavyweight, integer	n/a	168	168	n/a	168	168
BCC2 heavyweight, floating-point	n/a	168	168	n/a	168	168
ECC1 heavyweight, integer	n/a	n/a	n/a	200	200	200
ECC1 heavyweight, floating-point	n/a	n/a	n/a	200	200	200
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	200
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	200
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	112	112	112	112	112	112
BCC1 lightweight, floating-point	116	116	116	116	116	116
BCC1 heavyweight, integer	184	184	184	184	184	184
BCC1 heavyweight, floating-point	184	184	184	184	184	184
BCC2 lightweight, integer	n/a	116	116	n/a	116	116
BCC2 lightweight, floating-point	n/a	116	116	n/a	116	116
BCC2 heavyweight, integer	n/a	192	192	n/a	192	192
BCC2 heavyweight, floating-point	n/a	192	192	n/a	192	192
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

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	88	88	88	88	88	88
BCC1 lightweight, floating-point	92	92	92	92	92	92
BCC1 heavyweight, integer	160	160	160	160	160	160
BCC1 heavyweight, floating-point	160	160	160	160	160	160
BCC2 lightweight, integer	n/a	92	92	n/a	92	92
BCC2 lightweight, floating-point	n/a	92	92	n/a	92	92
BCC2 heavyweight, integer	n/a	168	168	n/a	168	168
BCC2 heavyweight, floating-point	n/a	168	168	n/a	168	168
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
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	112	112	112	112	112	112
BCC1 lightweight, floating-point	116	116	116	116	116	116
BCC1 heavyweight, integer	184	184	184	184	184	184
BCC1 heavyweight, floating-point	184	184	184	184	184	184
BCC2 lightweight, integer	n/a	116	116	n/a	116	116
BCC2 lightweight, floating-point	n/a	116	116	n/a	116	116
BCC2 heavyweight, integer	n/a	192	192	n/a	192	192
BCC2 heavyweight, floating-point	n/a	192	192	n/a	192	192
ECC1 heavyweight, integer	n/a	n/a	n/a	248	248	248
ECC1 heavyweight, floating-point	n/a	n/a	n/a	248	248	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

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for the SH2AWR supports two variations of the OSEK interrupt handling API calls. In addition to the API calls contained within the RTA-OSEK run-time libraries, inline versions are also supported. Using these inline versions will result in faster code. The inline versions of these API calls are all have the "os" prefix.

The inline API calls are restricted to the standard build applications that do not use RTA-TRACE. Inline calls contained within application code for other configurations will be automatically substituted with calls to the library API during compilation.

To take advantage of the inline versions in application code the substitutions in the following table should be used:

Library API call	Inline API call
`DisableAllInterrupts()`	`osDisableAllInterrupts()`
`EnableAllInterrupts()`	`osEnableAllInterrupts()`
`SuspendOSInterrupts()`	`osSuspendOSInterrupts()`
`ResumeOSInterrupts()`	`osResumeOSInterrupts()`
`SuspendAllInterrupts()`	`osSuspendAllInterrupts()`
`ResumeAllInterrupts()`	`osResumeAllInterrupts()`

6 Compatibility with v5.0.0

6.1 Updating the application version

To convert an existing v5.00 OIL configuration file to v5.0.1, load the file into the v5 RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.01. When the OIL configuration file is saved it will then use the v5.01 kernel libraries. This process can be reversed to move back to earlier kernel versions.

Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found at the front of this manual and on the ETAS Group website www.etasgroup.com.