RTA-OSEK

Binding Manual: SH2A/SHC

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: M00090-002

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/SHC 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	Renesas
Compiler	HEW4 SH Series C/C++ Compiler
Version	9.02.00.003

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.

This file must be compiled with the option `-cpu=SH2A` or `-cpu=SH2AFPU` as appropriate. See the example application for details of how this is done using the subcommand file `copts.sub` generated by the build script.

Important: When `osekdefs.c` is compiled the following two information messages may occur depending on the application configuration:

`C0005 (I) Precision lost` - The lost precision is a result of null pointer values. If the task concerned with the null pointer is reconfigured to use a different OSEK compliance class then a non-null value occurs.

`C0012 (I) Unused variable` - The unused variables relate to the variables passed to an empty function that is used in place of functions linked from the RTA-OSEK library. When the library versions are used the arguments are required.

The warnings are benign and build scripts generated by RTA-OSEK automatically suppress these warnings. The following compilation option should be used to suppress these warnings in user builds scripts:

```
-NOME=5,12
```

The RTA-OSEK Component has been compiled with the following options:

Option	Meaning
-cpu=SH2A	Compile for SH-2A CPUs
-OP=1	Turns on compiler optimization
-SIze	Optimize for size

2.2 Assembler

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

Vendor	Renesas
Assembler	HEW4 SH Series Assembler
Version	7.01.02.000

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

This file must be assembled with the option `-cpu=SH2A` or `-cpu=SH2AFPU` as appropriate. See the example application for details of how this is done using the subcommand file `aopts.sub` generated by the build script.

The RTA-OSEK Component has been assembled with the following options:

Option	Meaning
-cpu=SH2A	Assemble for the SH2A CPU

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
os_pur	RAM	RTA-OSEK zeroed data
POS	ROM	RTA-OSEK code
COS	ROM	RTA-OSEK read-only data
DOS	ROM	RTA-OSEK initialization data for ROS
BOS	RAM	RTA-OSEK zeroed data
ROS	RAM	RTA-OSEK initialized data
BOSTRACE	RAM	RTA-TRACE zeroed data

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

The startup file `cstart.src` supplied with the example program uses a suitably named section for the stack.

2.3.1 Sections

To make it easy to distinguish RTA-OSEK code and data from application code and data, RTA-OSEK does not use the default compiler generated sections. All RTA-OSEK sections are either prefixed "`os_`" or postfixed with "`OS`" or "`OSTRACE`".

When the RTA-OSEK and RTA-TRACE Component libraries were compiled the following command line option was used:

```
-SE=program=POS,const=COS,data=DOS,bss=BOS
```

Also the RTA-TRACE file `ostrace.c` contains the following directive enforcing the use of these sections:

```
#pragma section OSTRACE
```

As a result RTA-OSEK and RTA-TRACE uses sections as shown in the following table:

Default Section	RTA-OSEK Section	Section Contents
P	POS	Code
C	COS	Constant data
D	DOS	Initialized data (ROM copy)
R	ROS	Initialized data (RAM)
B	BOS BOSTRACE	Zeroed data

The sections `POS`, `COS`, `DOS`, `ROS`, `BOS` and `BOSTRACE` should be treated in the same way as their equivalent defaults in terms of linking.

RTA-OSEK also uses some additional RAM sections called `os_pur` and `os_pir`. These sections are initialized by the RTA-OSEK Component within

the StartOS() API call (unlike the standard C RAM variable sections DOS/ROS and BOS, which must be initialized in the application start-up code).

2.3.2 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 using the -cpu=SH2A compile option. 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.

2.3.3 Generating the standard library

Before the toolchain is used to build applications, the Standard Library Generator (lbgsh) must be run to create a standard library. This must be built with the -reent (Generate a re-entrant library) option, in addition to the options to match the users' application compilation process.

A re-entrant library requires some additional functions in order to link and work correctly. The compiler's manual states that these are errno_addr(), wait_sem() and signal_sem(). An example of errno_addr() is provided in example\target.c. The wait_sem() and signal_sem() functions should only be required if you use the malloc(), strtok() or file I/O library functions.

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 (.abs) 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/SHC. 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 Renesas

C compiler can generate appropriate interrupt handling code for a C function decorated with the `#pragma 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.src`.

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` 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 the macro 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):

```
-DEFINE=OS_IPL_INIT=1
```

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 `#pragma interrupt` function qualifier should also include the `resbank` interrupt specification (please refer to the Renesas compiler documentation for further details).

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 `–DEFINE=OS_ISR_FIX=1`.

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 option:

```
-DEF=OS_SECURITY_ID=1,OS_SECURITY_ID_VAL_0x60=H'1234aabb,OS_SECURITY_ID_VAL_0x64=H'ccdd8765
```

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()`.
`VBR`	See section 3.1.2

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): 88

Timing

API max usage (bytes): 88

Extended

API max usage (bytes): 84

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.

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 end of the section called `STACK`.

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>*`\shcsh2a\inc` directory. Note that `osfptgt.c` must first be compiled to an intermediate assembler file, which is then assembled to produce an object file. 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 not 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	166	166	172	246	298	304
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	238	238	244	318	374	400
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	290	342	296	370	422	376
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	62	62	62	62	62	62
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	62	62	62	62	62	62
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	82	82	82	82	82	82
Category 2 ISR, floating-point	RAM	84	84	84	84	84	84
	ROM	92	92	92	92	92	92
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	62	62	62	62	62	62
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	94	94	94	94	94	94
Category 2 ISR, floating-point	RAM	88	88	88	88	88	88
	ROM	104	104	104	104	104	104
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	66	66	66	66	66	66
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	144	192	232	158	206	264
	NS		122	170	210	128	176	242
	KL	2	68	122	156	82	128	188
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	26	26	26	26	26	26
ChainTask	SWL	1, 8	120	168	214	134	182	240
	SWH	1, 9	162	210	250	168	216	276
	NSL	8	120	168	214	134	182	240
	NSH	9	148	196	236	162	210	270
Schedule			106	106	124	106	106	124
GetTaskID			28	28	28	28	28	28
GetTaskState			88	88	88	100	100	100
EnableAllInterrupts			26	26	26	26	26	26
DisableAllInterrupts			34	34	34	34	34	34
ResumeAllInterrupts			36	36	36	36	36	36
SuspendAllInterrupts			54	54	54	54	54	54
ResumeOSInterrupts			46	46	46	46	46	46
SuspendOSInterrupts			64	64	64	64	64	64
GetResource	Task	7	24	24	28	24	24	28
	Combined	6	82	82	82	82	82	82
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	90	90	90	90	90	90
	Combined	6	156	156	156	156	156	156
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	138	138	216
	NS		n/a	n/a	n/a	104	104	194
	NS1i	10	n/a	n/a	n/a	66	n/a	n/a
	KL	2	n/a	n/a	n/a	66	66	144
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	60	60	60

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	248	248	480
	fp	11	n/a	n/a	n/a	284	284	560
	1i	10	n/a	n/a	n/a	20	n/a	n/a
GetAlarmBase			64	64	64	64	64	64
GetAlarm			104	104	104	104	104	104
SetRelAlarm			396	396	396	396	396	396
SetAbsAlarm			452	452	452	452	452	452
CancelAlarm			94	94	94	94	94	94
InitCounter			64	64	64	64	64	64
GetCounterValue			74	74	74	74	74	74
GetScheduleTableStatus		34	48	70	70	48	70	70
NextScheduleTable		34	208	162	162	220	162	162
StartScheduleTable		34	86	304	308	86	308	316
StopScheduleTable		34	64	90	90	64	90	90
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	12	12	12
ScheduleTable expiry point	Callback		6	6	6	6	6	6
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		30	30	30	30	30	30
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	30	30	30	30	30	30
Process container	Non-Yielding	33	16	16	16	16	16	16
osek_tick_alarm	<default>		90	90	90	90	90	90
	KL	2	38	38	38	38	38	38
osek_incr_counter			26	26	26	26	26	26
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			174	174	174	174	174	174
ShutdownOS	NoHook	12	40	40	40	40	40	40
	Hook	13	52	52	52	52	52	52
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			34	34	34	34	34	34
StopCOM			22	22	22	22	22	22
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	94	94	94	208	208	208

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	208	208	208	208	208	208
GetMessageResource			42	42	42	42	42	42
ReleaseMessageResource			36	36	36	36	36	36
GetMessageStatus			36	36	36	36	36	36
SendMessage	SW CCCA	1, 14	130	130	130	252	252	252
	SW CCCB	1, 15	242	242	242	252	252	252
	NS CCCA	14	130	130	130	252	252	252
	NS CCCB	15	242	242	242	252	252	252
	KL CCCA	2, 14	66	66	66	180	180	180
	KL CCCB	2, 15	166	166	166	180	180	180
main_dispatch	NoHook	12	122	122	154	122	122	154
	Hook	13	166	166	198	166	166	198
sub_dispatch	B1LF	19	32	32	32	32	32	32
	B1HI	20	92	92	92	92	92	92
	B1HF	21	108	108	108	108	108	108
	B2LI	22	n/a	82	106	n/a	82	106
	B2LF	23	n/a	92	116	n/a	92	116
	B2HI	24	n/a	266	336	n/a	266	336
	B2HF	25	n/a	288	352	n/a	288	352
	E1HI	26	n/a	n/a	n/a	386	386	456
	E1HF	27	n/a	n/a	n/a	402	402	472
	E2HI	28	n/a	n/a	n/a	n/a	n/a	456
	E2HF	29	n/a	n/a	n/a	n/a	n/a	472
ErrorHook support		16	40	40	40	40	40	40
	ServiceID	17	48	48	48	48	48	48
	Parameters	18	62	62	62	62	62	62
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	146	256	306	160	290	350
	NS		116	234	276	138	260	320
	KL	2	70	174	212	84	200	258
ChainTaskset	SWL	1, 8	124	236	286	138	262	322
	SWH	1, 9	178	292	336	184	318	372
	NSL	8	124	236	286	138	262	322
	NSH	9	164	286	330	178	304	366
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			62	62	62	62	62	62
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			60	60	60	60	60	60
TestSubTaskset			68	68	68	68	68	68
TestEquivalentTaskset			64	64	64	64	64	64
TickSchedule	SW	1	210	192	192	192	192	192
	NS		182	162	162	162	162	162
	KL	2	132	118	118	118	118	118
AdvanceSchedule	SW	1	192	182	182	182	182	182
	NS		164	144	144	144	144	144
	KL	2	122	102	102	102	102	102
StartSchedule			84	84	84	84	84	84
StopSchedule			70	70	70	70	70	70
GetScheduleStatus			100	100	100	100	100	100
GetScheduleValue			78	78	78	78	78	78
GetScheduleNext			10	10	10	10	10	10
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			34	34	34	34	34	34
GetExecutionTime			4	4	4	4	4	4
GetLargestExecutionTime			6	6	6	6	6	6
ResetLargestExecutionTime			4	4	4	4	4	4
GetStackOffset			24	24	24	24	24	24

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	144	192	232	158	206	264
	NS		122	170	210	128	176	242
	KL	2	68	122	156	82	128	188
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	26	26	26	26	26	26
ChainTask	SWL	1, 8	120	168	214	134	182	240
	SWH	1, 9	162	210	250	168	216	276
	NSL	8	120	168	214	134	182	240
	NSH	9	148	196	236	162	210	270
Schedule			126	126	140	126	126	140
GetTaskID			28	28	28	28	28	28
GetTaskState			88	88	88	100	100	100
EnableAllInterrupts			26	26	26	26	26	26
DisableAllInterrupts			34	34	34	34	34	34
ResumeAllInterrupts			36	36	36	36	36	36
SuspendAllInterrupts			54	54	54	54	54	54
ResumeOSInterrupts			46	46	46	46	46	46
SuspendOSInterrupts			64	64	64	64	64	64
GetResource	Task	7	24	24	28	24	24	28
	Combined	6	82	82	82	82	82	82
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	110	110	110	110	110	110
	Combined	6	192	192	192	192	192	192
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	138	138	216
	NS		n/a	n/a	n/a	104	104	194
	NS1i	10	n/a	n/a	n/a	66	n/a	n/a
	KL	2	n/a	n/a	n/a	66	66	144
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	60	60	60
GetEvent			n/a	n/a	n/a	10	10	10
WaitEvent	<default>		n/a	n/a	n/a	344	344	576
	fp	11	n/a	n/a	n/a	368	368	640

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	108	n/a	n/a
GetAlarmBase			64	64	64	64	64	64
GetAlarm			104	104	104	104	104	104
SetRelAlarm			396	396	396	396	396	396
SetAbsAlarm			452	452	452	452	452	452
CancelAlarm			94	94	94	94	94	94
InitCounter			64	64	64	64	64	64
GetCounterValue			74	74	74	74	74	74
GetScheduleTableStatus		34	48	70	70	48	70	70
NextScheduleTable		34	228	162	162	64	162	162
StartScheduleTable		34	86	312	128	86	320	128
StopScheduleTable		34	64	90	90	232	90	90
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	12	12	12
ScheduleTable expiry point	Callback		6	6	6	6	6	6
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		30	30	30	30	30	30
GetISRID		4	26	26	26	26	26	26
Process container	Yielding	32	30	30	30	30	30	202
Process container	Non-Yielding	33	16	16	16	16	16	16
osek_tick_alarm	<default>		90	90	90	90	90	90
	KL	2	38	38	38	38	38	38
osek_incr_counter			26	26	26	26	26	26
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			230	230	230	230	230	230
ShutdownOS	NoHook	12	40	40	40	40	40	40
	Hook	13	52	52	52	52	52	52
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			34	34	34	34	34	34
StopCOM			22	22	22	22	22	22
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	94	94	94	208	208	208
	CCCB	15	208	208	208	208	208	208
GetMessageResource			42	42	42	42	42	42
ReleaseMessageResource			36	36	36	36	36	36

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			36	36	36	36	36	36
SendMessage	SW CCCA	1, 14	130	130	130	252	252	252
	SW CCCB	1, 15	242	242	242	252	252	252
	NS CCCA	14	130	130	130	252	252	252
	NS CCCB	15	242	242	242	252	252	252
	KL CCCA	2, 14	66	66	66	180	180	180
	KL CCCB	2, 15	166	166	166	180	180	180
main_dispatch	NoHook	12	150	150	182	150	150	182
	Hook	13	192	192	224	192	192	224
sub_dispatch	B1LF	19	28	28	28	28	28	28
	B1HI	20	104	104	104	104	104	104
	B1HF	21	120	120	120	120	120	120
	B2LI	22	n/a	62	86	n/a	62	86
	B2LF	23	n/a	72	96	n/a	72	96
	B2HI	24	n/a	270	334	n/a	270	334
	B2HF	25	n/a	284	350	n/a	284	350
	E1HI	26	n/a	n/a	n/a	426	426	488
	E1HF	27	n/a	n/a	n/a	442	442	504
	E2HI	28	n/a	n/a	n/a	n/a	n/a	488
	E2HF	29	n/a	n/a	n/a	n/a	n/a	504
ErrorHook support		16	40	40	40	40	40	40
	ServiceID	17	48	48	48	48	48	48
	Parameters	18	62	62	62	62	62	62
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	82	82	82	82	82	82
Timing_termination		4	88	88	88	88	88	88
ActivateTaskset	SW	1	146	256	306	160	290	350
	NS		116	234	276	138	260	320
	KL	2	70	174	212	84	200	258
ChainTaskset	SWL	1, 8	124	236	286	138	262	322
	SWH	1, 9	178	292	336	184	318	372
	NSL	8	124	236	286	138	262	322
	NSH	9	164	286	330	178	304	366
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			62	62	62	62	62	62
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			60	60	60	60	60	60

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			68	68	68	68	68	68
TestEquivalentTaskset			64	64	64	64	64	64
TickSchedule	SW	1	210	192	192	192	192	192
	NS		182	162	162	162	162	162
	KL	2	132	118	118	118	118	118
AdvanceSchedule	SW	1	192	182	182	182	182	182
	NS		164	144	144	144	144	144
	KL	2	122	102	102	102	102	102
StartSchedule			84	84	84	84	84	84
StopSchedule			70	70	70	70	70	70
GetScheduleStatus			100	100	100	100	100	100
GetScheduleValue			78	78	78	78	78	78
GetScheduleNext			10	10	10	10	10	10
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			34	34	34	34	34	34
GetExecutionTime			114	114	114	114	114	114
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			10	10	10	10	10	10
GetStackOffset			24	24	24	24	24	24

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	242	294	334	252	300	356
	NS		270	316	356	282	330	386
	KL	2	146	198	232	156	204	262
TerminateTask	LExt	3	106	106	106	106	106	106

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	134	134	134	134	134	134
ChainTask	SWL	1, 8	258	312	354	272	326	386
	SWH	1, 9	304	358	392	316	364	422
	NSL	8	302	356	398	316	370	430
	NSH	9	340	388	428	352	402	462
Schedule			238	238	252	238	238	252
GetTaskID			46	46	46	46	46	46
GetTaskState			232	232	232	236	236	236
EnableAllInterrupts			42	42	42	42	42	42
DisableAllInterrupts			50	50	50	50	50	50
ResumeAllInterrupts			84	84	84	84	84	84
SuspendAllInterrupts			70	70	70	70	70	70
ResumeOSInterrupts			94	94	94	94	94	94
SuspendOSInterrupts			80	80	80	80	80	80
GetResource	Task	7	314	314	276	314	314	276
	Combined	6	302	302	302	302	302	302
	CLEx	3	290	290	290	290	290	290
ReleaseResource	Task	7	304	304	304	304	304	304
	Combined	6	386	386	386	386	386	386
	CLEx	3	264	264	264	264	264	264
SetEvent	SW	1	n/a	n/a	n/a	288	288	374
	NS		n/a	n/a	n/a	314	314	392
	NS1i	10	n/a	n/a	n/a	212	n/a	n/a
	KL	2	n/a	n/a	n/a	188	188	268
	KL1i	2, 10	n/a	n/a	n/a	140	n/a	n/a
ClearEvent			n/a	n/a	n/a	128	128	128
GetEvent			n/a	n/a	n/a	146	146	146
WaitEvent	<default>		n/a	n/a	n/a	470	470	684
	fp	11	n/a	n/a	n/a	478	478	740
	1i	10	n/a	n/a	n/a	200	n/a	n/a
GetAlarmBase			176	176	176	176	176	176
GetAlarm			180	180	180	180	180	180
SetRelAlarm			530	530	530	530	530	530
SetAbsAlarm			576	576	576	576	576	576
CancelAlarm			170	170	170	170	170	170
InitCounter			216	216	216	216	216	216
GetCounterValue			186	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	
GetScheduleTableStatus		34	62	84	84	62	84	84
NextScheduleTable		34	78	176	176	78	176	176
StartScheduleTable		34	100	326	142	100	334	142
StopScheduleTable		34	242	104	104	254	104	104
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	12	12	12
ScheduleTable expiry point	Callback		6	6	6	6	6	6
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		30	30	30	30	30	30
GetISRID		4	40	40	92	40	40	108
Process container	Yielding	32	30	30	30	30	30	202
Process container	Non-Yielding	33	16	16	16	16	16	16
osek_tick_alarm	<default>		128	128	128	128	128	128
	KL	2	38	38	38	38	38	38
osek_incr_counter			26	26	26	26	26	26
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			246	246	246	246	246	246
ShutdownOS	NoHook	12	54	54	54	54	54	54
	Hook	13	66	66	66	66	66	66
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			50	50	50	50	50	50
StopCOM			44	44	44	44	44	44
ReadFlag			28	28	28	28	28	28
ResetFlag			32	32	32	32	32	32
ReceiveMessage	CCCA	14	184	184	184	300	300	300
	CCCB	15	300	300	300	300	300	300
GetMessageResource			90	90	90	90	90	90
ReleaseMessageResource			92	92	92	92	92	92
GetMessageStatus			84	84	84	84	84	84
SendMessage	SW CCCA	1, 14	224	224	224	358	358	358
	SW CCCB	1, 15	342	342	342	358	358	358
	NS CCCA	14	224	224	224	358	358	358
	NS CCCB	15	342	342	342	358	358	358
	KL CCCA	2, 14	142	142	142	262	262	262
	KL CCCB	2, 15	246	246	246	262	262	262
main_dispatch	NoHook	12	150	150	182	150	150	182

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	192	192	224	192	192	224
sub_dispatch	B1LF	19	28	28	28	28	28	28
	B1HI	20	104	104	104	104	104	104
	B1HF	21	120	120	120	120	120	120
	B2LI	22	n/a	62	86	n/a	62	86
	B2LF	23	n/a	72	96	n/a	72	96
	B2HI	24	n/a	270	334	n/a	270	334
	B2HF	25	n/a	284	350	n/a	284	350
	E1HI	26	n/a	n/a	n/a	426	426	488
	E1HF	27	n/a	n/a	n/a	442	442	504
	E2HI	28	n/a	n/a	n/a	n/a	n/a	488
	E2HF	29	n/a	n/a	n/a	n/a	n/a	504
ErrorHook support		16	126	126	126	126	126	126
	ServiceID	17	132	132	132	132	132	132
	Parameters	18	158	158	158	158	158	158
validity_checks		3	22	22	22	22	22	22
Timing_dispatch		4	82	82	82	82	82	82
Timing_termination		4	88	88	88	88	88	88
ActivateTaskset	SW	1	326	386	432	346	412	480
	NS		354	414	460	374	440	508
	KL	2	218	274	312	232	302	352
ChainTaskset	SWL	1, 8	366	416	466	376	442	514
	SWH	1, 9	408	472	538	420	502	574
	NSL	8	412	460	526	420	490	556
	NSH	9	444	512	578	456	546	608
GetTasksetRef			114	114	114	114	114	114
MergeTaskset			264	264	264	264	264	264
AssignTaskset			166	166	166	166	166	166
RemoveTaskset			270	270	270	270	270	270
TestSubTaskset			294	294	294	294	294	294
TestEquivalentTaskset			290	290	290	290	290	290
TickSchedule	SW	1	334	310	310	310	310	310
	NS		348	350	350	350	350	350
	KL	2	236	218	218	218	218	218
AdvanceSchedule	SW	1	334	308	308	308	308	308
	NS		354	348	348	348	348	348
	KL	2	236	212	212	212	212	212

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			224	224	224	224	224	224
StopSchedule			180	180	180	180	180	180
GetScheduleStatus			218	218	218	218	218	218
GetScheduleValue			186	186	186	186	186	186
GetScheduleNext			84	84	84	84	84	84
SetScheduleNext			166	166	166	166	166	166
GetArrivalpointDelay			122	122	122	122	122	122
SetArrivalpointDelay			132	132	132	132	132	132
GetArrivalpointTasksetRef			116	116	116	116	116	116
GetArrivalpointNext			122	122	122	122	122	122
SetArrivalpointNext			174	174	174	174	174	174
TestArrivalpointWritable			128	128	128	128	128	128
GetExecutionTime			160	160	160	160	160	160
GetLargestExecutionTime			96	96	96	96	96	96
ResetLargestExecutionTime			92	92	92	92	92	92
GetStackOffset			24	24	24	24	24	24

Notes

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

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

4.2.4 Reserved Hardware Resources

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

4.3 Performance

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

4.3.1 Execution Times for RTA-OSEK API Calls

The following tables give the execution time (in CPU cycles) for each API call. (Note that: (1) the OSEK COM class was set to CCCA for systems without events and to CCCB for systems with events; (2) `ShutdownOS()` enters an

infinite loop; the execution time for `ShutdownOS()` reported below is the time up to the point at which `ShutdownOS()` calls `ShutdownHook()`).

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	103	129	163	109	121	175
	NS	91	115	149	91	113	169
	KL	55	83	115	57	79	133
TerminateTask	LExt	0	0	0	0	0	0
	H	157	159	157	157	161	157
ChainTask	SWL	305	359	415	349	381	441
	SWH	363	421	479	405	443	493
	NSL	303	359	417	349	381	437
	NSH	359	415	479	405	437	493
Schedule	SW	91	85	95	89	83	91
GetTaskID		25	33	29	25	27	27
GetTaskState		83	81	81	87	85	87
EnableAllInterrupts		21	25	27	21	21	21
DisableAllInterrupts		33	37	37	33	33	35
ResumeAllInterrupts		31	35	37	31	31	31
SuspendAllInterrupts		45	47	47	45	43	43
ResumeOSInterrupts		31	35	37	31	31	31
SuspendOSInterrupts		45	47	47	45	43	43
GetResource	Task	31	37	35	33	33	33
	Combined	57	57	57	55	57	57
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	79	79	81	81	77	81
	Combined	103	105	105	107	111	107
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	105	105	111
	NS	n/a	n/a	n/a	97	99	103
	KL	n/a	n/a	n/a	63	63	65
ClearEvent		n/a	n/a	n/a	59	55	57
GetEvent		n/a	n/a	n/a	25	23	23
WaitEvent	<default>	n/a	n/a	n/a	457	459	533

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	fp	n/a	n/a	n/a	479	483	551
GetAlarmBase		77	79	81	87	87	87
GetAlarm		99	103	103	101	103	101
SetRelAlarm		159	159	159	159	163	159
SetAbsAlarm		157	161	159	161	159	163
CancelAlarm		79	81	81	79	79	81
InitCounter		69	67	71	69	67	67
GetCounterValue		71	75	75	73	75	73
osek_tick_alarm	<default>	85	95	95	95	95	95
	KL	41	45	43	43	43	45
osek_incr_counter		19	21	21	21	21	21
GetActiveApplicationMode		15	15	15	17	17	17
StartOS		1065	1063	1063	1065	1063	1065
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	63	61	61	61	63	61
InitCOM		17	9	9	11	11	11
CloseCOM		9	9	9	13	11	13
StartCOM		45	51	51	125	129	125
StopCOM		21	21	21	25	25	25
ReadFlag		n/a	n/a	n/a	19	19	19
ResetFlag		n/a	n/a	n/a	13	13	13
ReceiveMessage		75	71	71	271	269	271
GetMessageResource		n/a	n/a	n/a	103	105	99
ReleaseMessageResource		n/a	n/a	n/a	133	131	131
GetMessageStatus		n/a	n/a	n/a	49	49	49
SendMessage	SW	199	223	259	403	419	479
	NS	189	211	247	391	409	471
	KL	105	129	161	271	291	349
ActivateTaskset	SW	99	433	475	99	457	487
	NS	79	413	455	81	437	463
	KL	41	403	409	51	369	457
	SW2	91	425	467	93	451	481
	NS2	79	413	455	81	437	463
	KL2	41	403	409	51	369	457
ChainTaskset	SWL	291	675	741	333	661	759
	SWH	361	739	803	395	721	815

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NSL	291	675	741	331	659	763
	NSH	357	731	797	393	717	807
GetTasksetRef		21	27	27	19	19	23
MergeTaskset		63	61	61	63	63	63
AssignTaskset		19	21	21	19	19	19
RemoveTaskset		65	65	65	65	65	67
TestSubTaskset		71	73	73	73	71	71
TestEquivalentTaskset		73	73	73	73	73	73
TickSchedule	SW	177	589	597	225	575	645
	NS	153	565	573	207	551	621
	KL	121	525	531	165	507	579
	SW2	173	581	587	225	547	627
	NS2	153	563	571	209	531	613
	KL2	119	523	531	167	489	571
AdvanceSchedule	SW	163	563	567	219	547	623
	NS	135	549	551	193	531	609
	KL	109	513	517	161	495	573
	SW2	157	563	569	213	533	615
	NS2	137	549	553	193	515	601
	KL2	111	513	519	161	479	563
StartSchedule		89	87	87	91	93	91
StopSchedule		81	75	75	81	79	83
GetScheduleStatus		95	89	93	95	95	95
GetScheduleValue		89	83	83	89	85	89
GetScheduleNext		23	21	21	23	23	23
SetScheduleNext		23	23	23	23	25	23
GetArrivalpointDelay		19	21	21	19	21	19
SetArrivalpointDelay		17	17	17	21	17	21
GetArrivalpointTasksetRef		15	15	15	15	15	15
GetArrivalpointNext		19	19	19	19	19	19
SetArrivalpointNext		19	19	19	19	19	19
TestArrivalpointWritable		25	23	23	27	25	27
GetExecutionTime		15	11	11	15	9	9
GetLargestExecutionTime		23	19	19	23	21	23
ResetLargestExecutionTime		19	19	19	19	21	19
GetStackOffset		27	27	27	27	27	27

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	109	127	159	109	125	177
	NS	97	115	147	95	115	167
	KL	63	83	115	61	79	133
TerminateTask	LExt	0	0	0	0	0	0
	H	427	431	441	439	435	437
ChainTask	SWL	621	681	739	673	703	775
	SWH	673	739	797	731	761	819
	NSL	623	685	739	673	703	767
	NSH	663	723	789	717	741	811
Schedule	SW	87	85	95	87	85	93
GetTaskID		31	31	31	31	31	25
GetTaskState		81	81	81	85	87	85
EnableAllInterrupts		23	23	23	23	23	23
DisableAllInterrupts		35	37	37	37	37	35
ResumeAllInterrupts		33	33	33	33	33	35
SuspendAllInterrupts		45	45	45	47	47	47
ResumeOSInterrupts		33	33	33	33	33	35
SuspendOSInterrupts		45	45	45	45	45	45
GetResource	Task	37	33	31	31	31	29
	Combined	57	53	57	59	61	59
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	79	83	81	81	81	81
	Combined	105	103	103	105	111	103
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	113	113	121
	NS	n/a	n/a	n/a	101	103	109
	KL	n/a	n/a	n/a	65	61	65
ClearEvent		n/a	n/a	n/a	65	65	65
GetEvent		n/a	n/a	n/a	27	27	27
WaitEvent	<default>	n/a	n/a	n/a	761	755	835
	fp	n/a	n/a	n/a	807	801	863
GetAlarmBase		79	79	81	79	83	79
GetAlarm		103	103	95	101	101	101

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		157	163	163	165	163	163
SetAbsAlarm		165	159	161	161	159	161
CancelAlarm		85	79	79	79	79	79
InitCounter		69	71	73	71	71	71
GetCounterValue		73	77	73	75	77	75
osek_tick_alarm	<default>	93	85	93	85	93	85
	KL	47	49	47	49	47	47
osek_incr_counter		17	23	25	29	29	29
GetActiveApplicationMode		9	9	9	9	9	9
StartOS		2975	2983	2979	2975	2979	2979
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	59	59	59	59	59	59
InitCOM		15	15	13	13	13	11
CloseCOM		13	13	15	11	11	11
StartCOM		55	49	49	119	125	123
StopCOM		29	25	23	23	23	23
ReadFlag		n/a	n/a	n/a	15	15	15
ResetFlag		n/a	n/a	n/a	17	17	17
ReceiveMessage		69	65	69	277	275	275
GetMessageResource		n/a	n/a	n/a	101	101	103
ReleaseMessageResource		n/a	n/a	n/a	135	135	133
GetMessageStatus		n/a	n/a	n/a	49	49	49
SendMessage	SW	209	225	259	405	415	471
	NS	199	213	247	393	409	461
	KL	109	133	165	271	285	341
ActivateTaskset	SW	93	425	465	99	453	485
	NS	81	411	451	85	441	465
	KL	45	407	409	49	373	455
	SW2	93	425	465	99	453	485
	NS2	81	411	451	85	441	465
	KL2	45	407	409	49	373	455
ChainTaskset	SWL	611	1003	1069	655	979	1097
	SWH	671	1053	1123	723	1037	1151
	NSL	609	993	1067	657	979	1101
	NSH	661	1031	1107	707	1021	1135
GetTasksetRef		25	25	21	21	21	27

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		71	65	67	67	67	69
AssignTaskset		21	31	25	23	17	17
RemoveTaskset		69	61	67	67	67	67
TestSubTaskset		83	75	75	81	79	79
TestEquivalentTaskset		75	79	79	71	71	71
TickSchedule	SW	175	579	583	221	561	635
	NS	153	557	563	205	545	619
	KL	123	523	523	169	505	579
	SW2	175	579	587	221	543	625
	NS2	153	557	565	205	527	611
	KL2	123	523	525	169	489	571
AdvanceSchedule	SW	157	567	567	211	547	623
	NS	135	547	547	193	531	607
	KL	109	513	515	155	495	571
	SW2	155	565	567	207	531	613
	NS2	133	545	547	191	513	599
	KL2	105	511	515	153	479	563
StartSchedule		99	95	95	95	93	93
StopSchedule		85	81	77	83	81	81
GetScheduleStatus		97	93	89	97	97	91
GetScheduleValue		93	83	83	87	85	89
GetScheduleNext		25	25	25	27	27	21
SetScheduleNext		23	21	27	21	21	21
GetArrivalpointDelay		25	23	21	25	25	25
SetArrivalpointDelay		27	23	23	29	29	29
GetArrivalpointTasksetRef		19	19	19	15	15	21
GetArrivalpointNext		23	15	21	23	23	23
SetArrivalpointNext		21	15	15	21	21	21
TestArrivalpointWritable		31	29	29	31	31	33
GetExecutionTime		161	155	159	165	171	161
GetLargestExecutionTime		33	33	31	31	31	33
ResetLargestExecutionTime		31	27	29	29	29	27
GetStackOffset		29	31	31	29	29	29

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	307	339	371	315	333	377
	NS	329	365	391	333	361	399
	KL	259	289	317	267	289	321
TerminateTask	LExt	469	487	489	479	479	483
	H	517	523	527	521	519	525
ChainTask	SWL	885	947	1009	935	965	1053
	SWH	931	991	1055	991	1009	1091
	NSL	909	985	1029	963	997	1071
	NSH	959	1023	1083	1023	1049	1121
Schedule	SW	149	147	161	149	151	159
GetTaskID		47	51	51	47	45	47
GetTaskState		321	327	329	327	327	327
EnableAllInterrupts		33	39	37	35	39	35
DisableAllInterrupts		41	49	61	45	45	53
ResumeAllInterrupts		53	59	61	57	57	57
SuspendAllInterrupts		53	61	65	59	59	59
ResumeOSInterrupts		53	59	61	57	57	57
SuspendOSInterrupts		53	61	65	59	59	59
GetResource	Task	459	465	267	503	499	303
	Combined	251	257	259	291	291	293
	CLEx	279	283	285	321	321	321
ReleaseResource	Task	257	259	261	295	297	297
	Combined	263	271	271	299	299	303
	CLEx	233	237	235	273	271	273
SetEvent	SW	n/a	n/a	n/a	341	339	339
	NS	n/a	n/a	n/a	355	355	359
	KL	n/a	n/a	n/a	315	317	319
ClearEvent		n/a	n/a	n/a	107	105	105
GetEvent		n/a	n/a	n/a	283	285	283
WaitEvent	<default>	n/a	n/a	n/a	873	875	947
	fp	n/a	n/a	n/a	911	905	975
GetAlarmBase		217	223	233	227	227	237
GetAlarm		233	239	247	237	241	251

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		305	313	321	315	313	327
SetAbsAlarm		295	305	317	305	301	317
CancelAlarm		219	223	235	229	227	237
InitCounter		319	323	341	327	329	341
GetCounterValue		207	209	211	209	217	209
osek_tick_alarm	<default>	117	121	117	127	127	127
	KL	43	51	49	53	55	55
osek_incr_counter		21	29	29	31	31	31
GetActiveApplicationMode		9	9	9	13	13	13
StartOS		3079	3085	3085	3081	3079	3079
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	55	57	57	59	59	59
InitCOM		13	15	15	17	17	17
CloseCOM		11	17	17	17	17	19
StartCOM		73	71	69	149	149	147
StopCOM		33	41	41	41	41	43
ReadFlag		n/a	n/a	n/a	51	47	47
ResetFlag		n/a	n/a	n/a	43	45	45
ReceiveMessage		185	191	191	391	389	389
GetMessageResource		n/a	n/a	n/a	427	431	431
ReleaseMessageResource		n/a	n/a	n/a	425	423	423
GetMessageStatus		n/a	n/a	n/a	151	149	151
SendMessage	SW	503	539	563	707	723	769
	NS	527	565	587	725	755	799
	KL	401	433	461	575	593	629
ActivateTaskset	SW	489	821	881	503	875	893
	NS	535	901	907	545	829	913
	KL	459	819	823	445	759	799
	SW2	489	821	881	503	875	893
	NS2	535	901	907	545	829	913
	KL2	459	819	823	445	759	799
ChainTaskset	SWL	1105	1475	1537	1123	1421	1547
	SWH	1107	1497	1559	1167	1503	1597
	NSL	1111	1481	1581	1177	1497	1581
	NSH	1133	1521	1625	1195	1529	1597
GetTasksetRef		255	259	261	255	257	255

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		157	161	165	167	165	167
AssignTaskset		99	103	107	103	107	103
RemoveTaskset		161	167	167	163	165	163
TestSubTaskset		175	183	181	179	181	179
TestEquivalentTaskset		173	175	177	181	181	181
TickSchedule	SW	237	1031	1035	655	989	1023
	NS	249	1055	1059	677	1013	1047
	KL	175	975	979	595	933	967
	SW2	235	1031	1031	655	971	1007
	NS2	249	1055	1055	677	995	1031
	KL2	175	975	975	595	913	951
AdvanceSchedule	SW	229	1019	1021	647	981	1015
	NS	249	1051	1051	671	1009	1043
	KL	179	971	969	591	929	963
	SW2	229	1019	1017	647	963	999
	NS2	249	1051	1049	671	991	1027
	KL2	179	971	965	591	909	945
StartSchedule		153	159	155	167	159	161
StopSchedule		125	129	129	133	129	131
GetScheduleStatus		145	149	149	149	147	149
GetScheduleValue		131	137	135	139	137	139
GetScheduleNext		57	59	59	65	65	67
SetScheduleNext		103	107	105	103	101	105
GetArrivalpointDelay		69	79	79	75	79	77
SetArrivalpointDelay		91	91	91	101	101	99
GetArrivalpointTasksetRef		59	63	63	71	67	69
GetArrivalpointNext		65	69	69	63	63	67
SetArrivalpointNext		109	111	111	103	109	105
TestArrivalpointWritable		69	69	69	79	77	83
GetExecutionTime		191	197	197	189	195	199
GetLargestExecutionTime		239	243	241	241	241	243
ResetLargestExecutionTime		229	229	231	231	233	231
GetStackOffset		29	33	37	33	33	33

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	29	29	29	29	29	29
	Cat 2	41	153	155	155	155	155

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	29	29	29	29	29	29
	Cat 2	293	401	399	397	397	401

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	29	31	31	31	31	31
	Cat 2	287	399	395	393	397	397

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.

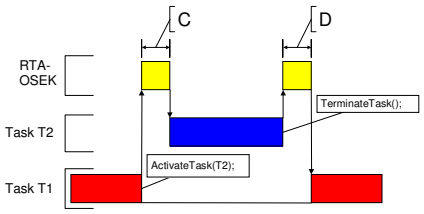

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

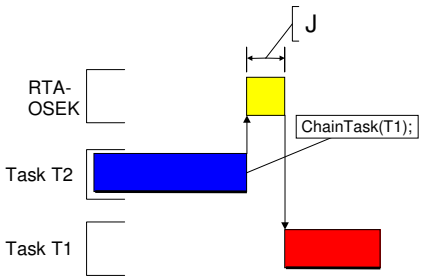

Figure 2: Task Chaining

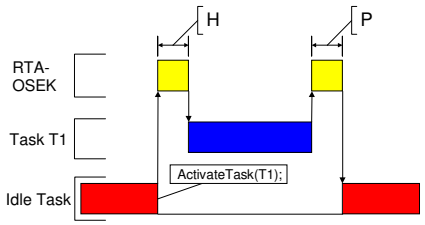

Figure 3: Task Activation from Idle Task

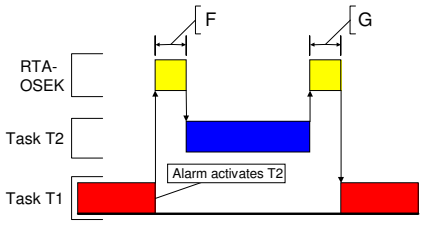

Figure 4: Task Activation from an Alarm

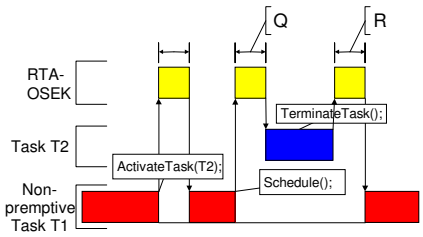

Figure 5: Non-Premptive Task Calls Schedule()

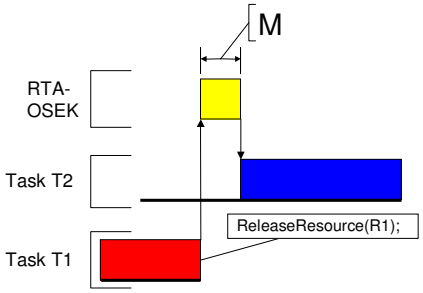

Figure 6: Blocked Task Activated by ReleaseResource()

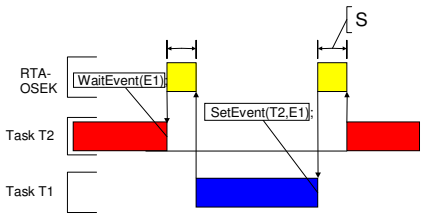

Figure 7: Waiting Task Activated by SetEvent()

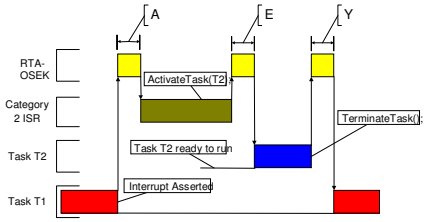

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	91	139	155	89	137	153
Figure 1: D	Heavy, Basic/Extended	155	201	219	203	201	221
ChainTask	Light, Basic	215	267	321	223	267	335
Figure 2: J	Heavy, Basic/Extended	517	619	689	571	617	695
Pre-emption	Light, Basic	207	249	321	207	249	333
Figure 1: C	Heavy, Basic/Extended	299	343	409	337	353	435
From idle task	Light, Basic	207	249	321	211	251	335
Figure 3: H	Heavy, Basic/Extended	299	343	409	339	355	437
Triggered by alarm	Light, Basic	315	353	429	313	353	439
Figure 4: F	Heavy, Basic/Extended	407	445	513	445	461	543
Schedule	Light, Basic	177	197	243	177	197	235
Figure 5: Q	Heavy, Basic/Extended	269	289	331	305	307	343
Release resource	Light, Basic	191	211	249	191	209	241
Figure 6: M	Heavy, Basic/Extended	283	303	337	319	321	349
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	529	533	675
From category 2 ISR	Light, Basic	227	317	351	295	315	341
Figure 8: E	Heavy, Basic/Extended	319	409	439	423	427	449

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	369	411	435	371	411	433
Figure 1: D	Heavy, Basic/Extended	429	469	493	475	469	491
ChainTask	Light, Basic	533	585	645	541	575	661
Figure 2: J	Heavy, Basic/Extended	1103	1199	1281	1157	1189	1285
Pre-emption	Light, Basic	425	459	529	425	463	541
Figure 1: C	Heavy, Basic/Extended	515	553	623	557	577	665
From idle task	Light, Basic	427	459	529	425	463	541
Figure 3: H	Heavy, Basic/Extended	517	553	623	557	577	665
Triggered by alarm	Light, Basic	533	567	639	529	569	647
Figure 4: F	Heavy, Basic/Extended	625	665	731	665	683	771
Schedule	Light, Basic	395	405	455	395	407	451
Figure 5: Q	Heavy, Basic/Extended	485	499	549	527	529	575
Release resource	Light, Basic	409	423	461	413	423	459
Figure 6: M	Heavy, Basic/Extended	499	517	555	545	545	583
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	757	757	901
From category 2 ISR	Light, Basic	729	813	857	809	817	851
Figure 8: E	Heavy, Basic/Extended	819	907	951	941	939	975

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	477	519	543	475	511	537
Figure 1: D	Heavy, Basic/Extended	519	561	577	553	557	579
ChainTask	Light, Basic	799	853	921	801	851	939
Figure 2: J	Heavy, Basic/Extended	1455	1549	1637	1499	1537	1637
Pre-emption	Light, Basic	609	649	723	619	655	727
Figure 1: C	Heavy, Basic/Extended	695	747	815	753	769	851
From idle task	Light, Basic	609	653	729	623	657	731
Figure 3: H	Heavy, Basic/Extended	695	751	821	755	771	855
Triggered by alarm	Light, Basic	739	781	853	751	791	861
Figure 4: F	Heavy, Basic/Extended	825	877	945	885	903	983
Schedule	Light, Basic	439	449	503	441	451	501
Figure 5: Q	Heavy, Basic/Extended	525	547	595	575	573	623
Release resource	Light, Basic	569	577	619	605	613	655
Figure 6: M	Heavy, Basic/Extended	655	675	711	739	735	777
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1007	995	1135
From category 2 ISR	Light, Basic	757	839	885	829	835	879
Figure 8: E	Heavy, Basic/Extended	843	937	977	963	957	1001

4.4 Configuration of Run-time Context

The run-time contexts of all tasks reside on the same stack and are recovered when the task terminates. As a result, run-time contexts of mutually exclusive tasks are effectively overlaid. The RTA-OSEK GUI is able to calculate the worst-case stack requirement for the entire application, based on the declared stack usage, the priorities and the resource occupation of individual tasks.

The size of the run-time context of a task depends on the task type and the system configuration. The following tables give the sizes (in bytes) for different OS status and configurations:

Standard

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Pre- and Post-Task hooks not used						
Task type						
BCC1 lightweight, integer	72	72	72	72	72	72
BCC1 lightweight, floating-point	76	76	76	76	76	76
BCC1 heavyweight, integer	128	128	128	128	128	128
BCC1 heavyweight, floating-point	128	128	128	128	128	128
BCC2 lightweight, integer	n/a	76	76	n/a	76	76
BCC2 lightweight, floating-point	n/a	76	76	n/a	76	76
BCC2 heavyweight, integer	n/a	132	136	n/a	132	136
BCC2 heavyweight, floating-point	n/a	132	136	n/a	132	136
ECC1 heavyweight, integer	n/a	n/a	n/a	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	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	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	184
ECC1 heavyweight, floating-point	n/a	n/a	n/a	184	184	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

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	80	80	80	80
BCC1 lightweight, floating-point	84	84	84	84	84	84
BCC1 heavyweight, integer	132	132	132	132	132	132
BCC1 heavyweight, floating-point	132	132	132	132	132	132
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	140	140	n/a	140	140
BCC2 heavyweight, floating-point	n/a	140	140	n/a	140	140
ECC1 heavyweight, integer	n/a	n/a	n/a	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	204
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	204
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	84	84	84	84	84	84
BCC1 lightweight, floating-point	88	88	88	88	88	88
BCC1 heavyweight, integer	136	136	136	136	136	136
BCC1 heavyweight, floating-point	136	136	136	136	136	136
BCC2 lightweight, integer	n/a	88	88	n/a	88	88
BCC2 lightweight, floating-point	n/a	88	88	n/a	88	88
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	204	204	204
ECC1 heavyweight, floating-point	n/a	n/a	n/a	204	204	204
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

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	80	80	80	80
BCC1 lightweight, floating-point	84	84	84	84	84	84
BCC1 heavyweight, integer	132	132	132	132	132	132
BCC1 heavyweight, floating-point	132	132	132	132	132	132
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	140	140	n/a	140	140
BCC2 heavyweight, floating-point	n/a	140	140	n/a	140	140
ECC1 heavyweight, integer	n/a	n/a	n/a	212	212	212
ECC1 heavyweight, floating-point	n/a	n/a	n/a	212	212	212
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	216
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	216
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	84	84	84	84	84	84
BCC1 lightweight, floating-point	88	88	88	88	88	88
BCC1 heavyweight, integer	136	136	136	136	136	136
BCC1 heavyweight, floating-point	136	136	136	136	136	136
BCC2 lightweight, integer	n/a	88	88	n/a	88	88
BCC2 lightweight, floating-point	n/a	88	88	n/a	88	88
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	216	216	216
ECC1 heavyweight, floating-point	n/a	n/a	n/a	216	216	216
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	220
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	220

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for the SH2A/SHC 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 Pre-v5.00 Kernels

6.1 Updating the Application Version

To convert an existing OIL configuration file that uses the v4.00 kernel library to use the v5.00 kernel libraries, load the file into the RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.00. When the OIL configuration file is saved it will then use the v5.00 kernel libraries. This process can be reversed to move back to earlier kernel versions.

6.2 Changes between versions 4.00 and 5.00

The following modifications have been made between versions 4.00 and 5.00:

- The compiler version used to build and test the libraries has been modified updated

V4.00 tools	v5.00 tools
Renesas HEW V4.01 (tools v9.00 release 04A) SH SERIES C/C++ Compiler V.9.00.03.006 SH SERIES ASSEMBLER V.7.01.01.000 Optimizing Linkage Editor V.9.01.00.003 Standard library generator V.3.00.02.001	Renesas HEW V4.02 (tools v9.01 release 01) SH Series C/C++ Compiler v9.01.01.000 SH Series Assembler v7.01.01.000 Optimizing Linkage Editor v9.03.00.001 Standard library generator v3.00.02.002

- 32-bit timers are now supported
- Addition of the inline interrupt control APIs
- Support the SH7254x CPU family (i.e. SH72513 and SH72546)
- Support for the FLASH key code at location 0x60 to 0x67
- Support to reduce the vector table size to only contain the used vectors
- ORTI debugging is now supported for either the Lauterbach Trace32 debugger or a generic ORTI debugger
- The Floating-point wrappers now preserve the state of the FP exceptions in the FPSCR

6.3 Changes between versions 5.00 and 5.05

The following modifications have been made between versions 5.00 and 5.05:

- The compiler version used to build and test the libraries has been modified updated

V5.00 tools	v5.05 tools
Renesas HEW V4.02 (tools v9.01 release 01) SH Series C/C++ Compiler v9.01.01.000 SH Series Assembler v7.01.01.000 Optimizing Linkage Editor v9.03.00.001 Standard library generator v3.00.02.002	Renesas HEW V4.04.01.001 (tools v9.20 release 00) SH Series C/C++ Compiler v9.02.00.003 SH Series Assembler v7.01.02.000 Optimizing Linkage Editor v9.04.00.000 Standard library generator v3.00.02.003

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.