RTA-OSEK

Binding Manual: HC12X16/Metrowerks

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 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 K.K. Yokohama 220-6217, Japan Tel.: +81 45 222-0900 Fax: +81 45 222-0956 sales.jp@etas.com	**ETAS Ltd.** Derby DE21 4SU, UK Tel.: +44 1332 253770 Fax: +44 1332 253779 sales.uk@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 (Shanghai) Co., Ltd.** Shanghai 200120, P.R. China Tel.: +86 21 5037 2220 Fax: +86 21 5037 2221 sales.cn@etas.com
ETAS in Italy 10135 TORINO, Italy Tel.: +39 011 3285 988 Fax: +39 (011) 3285 256 sales.it@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
ETAS in Brazil CEP-05802-140 São Paulo, Brazil Tel.: +55 11 2162-0252 sales.br@etas.com	**ETAS in Russia** Moscow, 129515, Russia Tel.: +7 495 937 0400 998 sales.ru@etas.com

Copyright Notice

Version: M00077-004

Disclaimer

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

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

Trademarks

RTA-OSEK and RTA-TRACE are trademarks of ETAS GmbH.

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Continental Automotive GmbH.

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 HC12X16/Metrowerks port of ETAS' 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 ETAS to confirm whether or not this is possible.

2.1 Memory Model

The S12X architecture offers three memory models: small, banked, and large. **This port is built for the large model.**

2.1.1 Background

The S12X has a 16-bit logical address space. Standard S12X instructions always use logical addresses. Within this logical address space there are 3 banked memory-windows: one for Flash memory at addresses `0x8000` to `0xBFFF`, one for RAM at addresses `0x1000` to `0x1FFF` and one for EEPROM at addresses `0x0800` to `0x0BFF`. Whenever a standard S12X instruction references memory in the Flash banked memory-window the contents of the `PPAGE` register are used to determine which of several Flash memory pages is currently visible in the memory-window. Likewise the `RPAGE` and `EPAGE` registers are used for the RAM and EEPROM banked memory-windows respectively.

Placement of code and data into memory pages is controlled by the Freescale linker file. See the "Segmentation" section of the "Freescale HC12 Back End" chapter of the Freescale Compiler Manual for details.

In addition to its 16-bit logical address space the S12X also has a 23-bit global address space. This global address space contains all physical memory. That is, all of the Flash, RAM and EEPROM pages as well as the unbanked memory are simultaneously visible in the global address space. The most significant 7 bits of a global address are taken from the `GPAGE` register and the least significant 16 bits come from the address field of an extended `Gxxxx` instruction (e.g. `GLDAA` or `GSTAA`). The Freescale linker uses information in the linker file to generate 23 bit global addresses.

There is a good description of how memory works in the S12X in the Freescale library source file

```
<install>\lib\hc12c\src\DATAPAGE.C
```

Where `<install>` is the root of the Codewarrior toolchain installation.

Note: when this manual refers to "unbanked ROM" it means Flash memory **not** in the `0x8000` to `0xBFFF` banked memory-window. Likewise when this

manual refers to "unbanked RAM" it means RAM **not** in the 0x1000 to 0x1FFF banked memory-window.

2.1.2 Function Addresses

In the large memory model functions have 3-byte addresses, with the upper byte coming from the PPAGE register. The most significant byte indicates the page of Flash that should be mapped into the banked memory-window between 0x8000 and 0xBFFF. The currently-mapped page is stored in the PPAGE register and the assembly instructions CALL and RTC maintain this implicitly.

"Near" functions have only 2-byte addresses, and must be present in the 16-bit logical address space at the time of calling. Near functions must be placed in an unbanked memory-area.

2.1.3 Data Addresses

In the large memory model, data is "far" by default, and pointers to data are3-bytes long. "Near" data only has a 2-byte address and consequently pointers to near data are 2 bytes. Near data must be placed in an unbanked memory area.

The internal data structures used by RTA-OSEK are 'near', and must be placed in unbanked memory.

2.1.4 Preservation of PAGE Registers by interrupt handlers

In the large memory model, it is necessary to preserve the PPAGE, EPAGE, GPAGE and RPAGE registers during interrupt servicing.

The PPAGE register is preserved by the CALL/RTC matching. The other registers must be preserved by the service routine.

In RTA-OSEK, GPAGE and RPAGE are preserved by the Category 2 ISR mechanism, and by Category 1 ISRs using the "__interrupt" keyword.

2.1.5 Additional Preservation of PAGE Registers

Modification of the EPAGE register within an application is supported by RTA-OSEK with the floating point wrappers. These wrappers are found in the files osfptgt.c and osfptgt.h in the <RTA-OSEK install dir>\HC12XMW16\inc folder. To apply this additional context saving by the wrappers all tasks and ISRs that modify the EPAGE register should be marked as using floating point in the RTA-OSEK GUI. Further details on the floating point wrappers can be found in the RTA-OSEK User Guide.

2.2 Compiler

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

Vendor	Freescale CodeWarrior
Compiler	chc12.exe
Version	5.0.38 Build 9056

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

Option	Description
`-Ml`	Build for the large memory model

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
`-Ml`	Build for the large memory model
`-CpuHCS12X`	Use extended instruction set. Without this, RTA-OSEK's stack figures may be incorrect.
`-CpRPAGE`	Preserve RPAGE register in `__interrupt` functions
`-CpGPAGE`	Preserve GPAGE register in `__interrupt` functions

Note: When compiling the automatically generated file `osekdefs.c` you may see warnings saying that functions defined in `osekdefs.c` were previously declared in different segments. This is an artifact of the way that `osekdefs.c` is generated and the warnings may be ignored.

Note: When compiling the automatically generated RTA-TRACE file `ostrace.c` you may see warnings saying that variables defined in `ostrace.c` were previously declared in different segments. This is an artifact of the way that `ostrace.c` is generated and the warnings may be ignored.

2.3 Assembler

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

Vendor	Freescale CodeWarrior
Assembler	ahc12.exe
Version	5.0.30 Build 9056

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

Option	Description
`-Ml`	Build for the large memory model

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

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

Option	Description
`-Ml`	Build for the large memory model
`-CpuHCS12X`	Enable extended instruction set

2.4 Linker/Locator

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

Option	Description
`-FE`	Only ELF/DWARF format supported

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

Sections	ROM/RAM	Description
`os_pid`	Unbanked ROM	RTA-OSEK read-only data
`os_pird`	Unbanked ROM	RTA-OSEK initialization data
`os_intvec`	Unbanked ROM	Vector table if generated by RTA-OSEK GUI
`os_intvechi`	Unbanked ROM	Non-relocatable vectors, if vectors generated by RTA-OSEK
`os_pir`	Unbanked RAM	RTA-OSEK initialized data
`os_pur`	Unbanked RAM	RTA-OSEK uninitialized data
`os_data_unbanked`	Unbanked RAM	RTA-OSEK data
`os_constdata_unbanked`	Unbanked ROM	RTA-OSEK read-only data
`os_text_unbanked`	Unbanked ROM	RTA-OSEK interrupt handling code
`os_text`	ROM	RTA-OSEK code

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

C Library Functions	Description
_COPY	Structure copying routine
_FPCMP	Far pointer comparison
_LCMP	Long compare
_CONV_GLOBAL_TO_NEAR	Global to near pointer conversion
_CONV_NEAR_TO_GLOBAL	Near to global pointer conversion

This port of RTA-OSEK is built for the `ansixl.lib`, `ansixli.lib` or `ansixlf.lib` runtime libraries. The compiler flag `-Ml` specifies the large memory model. It is a linker error to attempt to mix modules built for different memory models.

The pointer conversion routines listed above are provided as source code in the `DATAPAGE.C` file shipped with CiodeWarrior.

2.4.1 Section Placement

The vector table must be located in **unbanked** memory. If the vector table is generated by RTA-OSEK then it is in sections `os_intvec` and `os_intvechi`, which must be located in unbanked memory with the `PLACEMENT` clause of the linker file.

Some of the RTA-OSEK code concerned with interrupt handling must be loaded into unbanked memory. This code is in section `os_text_unbanked`, which should appear in the `PLACEMENT` section of the linker file and be mapped to unbanked memory. E.g. the unbanked flash areas `0x4000..0x7FFF` or `0xC000..0xFF0F`.

All other RTA-OSEK code is in the `os_text` section. This section may be placed in banked or unbanked memory.

The kernel expects to find its internal read-only variables in near memory. Read-only variables are in the `os_pid`, `os_pird` and `os_constdata_unbanked` sections, which should appear in the `PLACEMENT` section of the linker file and be mapped to unbanked memory. E.g. the unbanked flash areas `0x4000..0x7FFF` or `0xC000..0xFF0F`.

The kernel expects to find its writeable internal variables in near memory. Writeable variables are in the `os_pir`, `os_pur` and `os_data_unbanked` sections, which should appear in the `PLACEMENT` section of the linker file and be mapped to the unbanked RAM area `0x2000..0x3FFF`.

The linker will omit code that is not referenced by other code unless it appears in the `ENTRIES` section in the linker file. Thus, `os_intvec` and `os_intvechi` must appear in the `ENTRIES` section if used.

2.4.2 Pointers Passed to RTA-OSEK API Functions

Note: All RTA-OSEK API pointer arguments referring to OS objects are near (i.e. they are decorated with the `__near` modifier). This means that if you call an RTA-OSEK API function `GetAlarmBase(AlarmType, AlarmBaseRefType)` then the `AlarmBaseType` object pointed to by the `AlarmBaseRefType` type argument must be in near memory. Objects allocated on the stack are always in near memory. To place a global object in near memory use `#pragma DATA_SEG` or `#pragma CONST_SEG` with a `__NEAR_SEG` modifier. See the "Segmentation" section of the "Freescale HC12 Back End" chapter of the Freescale Compiler Manual. The section declared with the pragma must then be placed in unbanked memory by the linker file.

Any objects created by RTA-OSEK will be automatically placed in near memory. To ensure that more efficient near accesses (i.e. accesses that do not need to load the `GPAGE` register) are made to objects created by RTA-OSEK the automatically generated header files included by application source code (e.g. `osek.h` and `oseklib.h`) start with the code:

```
#pragma push
#pragma CONST_SEG __NEAR_SEG os_constdata_unbanked
#pragma DATA_SEG __NEAR_SEG os_data_unbanked
```

and end with the code:

```
#pragma pop
```

This code informs the compiler that all objects declared in the header file (or any nested header files) are in the near sections `os_constdata_unbanked` or `os_data_unbanked` and can be accessed with near memory references.

As a result of the above, if you re-declare an object that is declared in one of the automatically generated header files (e.g. using `DeclareTask()`) you may get a compiler warning saying that the object was previously declared in a different segment. These warnings are benign. Application code will generate far references to the near data. This is inefficient, but works. To avoid the warnings and inefficiencies there are two options.

- Do not re-declare objects declared inside the header files automatically generated by RTA-OSEK.
- If you do re-declare objects, use code like the following :

```
#pragma push
#pragma CONST_SEG __NEAR_SEG os_constdata_unbanked
#pragma DATA_SEG __NEAR_SEG os_data_unbanked
```

```
/* Re-declaration. */

#pragma pop
```

Pointers used for 'output' parameters of API (eg, `GetTaskID(TaskRefType TaskID)`) are global, and may refer to any read/write memory.

2.5 Debugger

ORTI is the OSEK Run-Time Interface that is supported by RTA-OSEK. Support is provided for the debuggers in the following table. Further information about ORTI for RTA-OSEK can be found in the *RTA-OSEK ORTI Guide*.

ORTI Compatible Debuggers
iSYSTEM winIDEA
Lauterbach TRACE32

2.5.1 Lauterbach TRACE32

The RTA-OSEK GUI produces a file with the extension `.ort`. This file should be loaded into the TRACE32 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.

2.5.2 iSYSTEM winIDEA

The evaluation of several ORTI values in the winIDEA debugger relies on the presence of several pointers which are added to the bottom of `osekdefs.c`. Since these pointers are not referenced anywhere else in an application, CodeWarrior's SmartLinker will strip them by default. To avoid this, force the linker to link everything from `osekdefs.c` by adding the following command to the linker parameter (`.prm`) file:

```
ENTRIES
    osekdefs.o:*
END
```

It may be helpful to refer to the supplied example application's linker parameter file `hcs12x.prm`, which includes the above `ENTRIES` command.

The pointers and the dummy function in `osekdefs.c` will not be generated when the debugger type is set to `<none>` in the RTA-OSEK GUI. When debugger support is not required it may also be desirable to remove the above `ENTRIES` command from the linker parameter file, to avoid linking unused objects.

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for HC12X16/Metrowerks. 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 *MC9S12XEP100 Data Sheet*.

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

IPL Value	CCRW	Description
0	CCR.I = 0; CCRH = 0	User level
1..7	CCR.I = 0; CCRH = [1..7]	Category 1 and 2 interrupts
8	CCR.I = 1; CCRH = any	Non-maskable 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
Fixed vectors 0xFFFA, 0xFFFC	Non-maskable interrupts: must be defined as Category 1 and have IPL of 8
Relocatable vectors 0xF4 to 0xF8	Non-maskable interrupts: must be defined as Category 1 and have IPL of 8
Relocatable vectors 0x60 to 0xF3	Priority 1..7; may be defined as category 1 or 2.
Relocatable vector 0x10	Non-maskable interrupt: must be defined as Category 1 and have IPL of 8

The valid base addresses for the vector table are:

Base Register	Notes
0xFF	Default base address
0x40..0x7F, 0xC0..0xFE	Unbanked Flash
0x80..0xBF	Banked Flash. Only valid if application does not change PPAGE.
0x08..0x0B, 0x10..0x3F	Banked EEPROM; banked and unbanked SRAM. Not recommended. User is responsible for setup and for managing side-effects.
0x0C..0x0F	Un**bank**ed EEPROM. Not recommended.

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 Freescale CodeWarrior C compiler can generate appropriate interrupt handling code for a C function decorated with the `__interrupt` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

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

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 (xx represents the 2 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
IVBR + *xx*	os_wrapper_*xx*

The S12X has some vectors that have a fixed location (vectors 0xFFFC, 0xFFFA) and the rest of the vectors are relocatable. If a vector table is generated it is in two sections. Section os_intvechi contains the fixed location vectors and must be located at 0xFFFA. Section os_intvec contains the relocatable vectors and can be placed at any of the locations outlined in the processor documentation. The user is responsible for initializing the Interrupt Vector Base Register (IVBR).

3.1.6 Interrupt Priority Levels

The priority at which a hardware interrupt is taken is set in the INT_CFDATA registers under the control of the INT_CFADDR register.

The RTA-OSEK GUI generates a table, called os_InitIrqLevels, which must be used to initialize the INT_CFDATA registers. This table contains the priority levels for interrupts defined in the application.

Important: The os_InitIrqLevels table must be copied to the INT_CFDATA registers before the call to StartOS() otherwise interrupts will not work correctly.

The init_target() function in target.c in the example application, located in <RTA-OSEK install directory>\HC12XMW16\Example\, gives an example of how to copy os_InitIrqLevels to the correct location.

3.2 Register Settings

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

Register	Value	Notes
CCRH	os_oim	Set the IPL to block out all category 2 interrupts
CCR.I	0	clear the I bit in CCR (e.g. by issuing __asm (cli);)
INT_CFDATA*x*	*y*	Initialize interrupt priority table

The RTA-OSEK Component does not reserve the use of any hardware registers.

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 short`.

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): 42

Timing

API max usage (bytes): 42

Extended

API max usage (bytes): 42

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.

3.3.3 Initialization of the stack pointer

The S12X instructions that push data onto the stack decrement the `SP` register before using it. Therefore the start-up code provided with the compiler initializes the `SP` to the value of `__SEG_END_SSTACK`, which is a linker-defined constant equal to the address of the first byte beyond the stack area.

For example, if the linker command file specifies that the `SSTACK` section should be placed in memory at addresses `0x3000-0x3FFF`, then the `SP` will be initialized to `0x4000`. A subsequent stack 'push' will therefore write to memory at `0x3FFF` (and not `0x4000`).

3.4 User vs. Supervisor State

The latest revision of the CPU ('S12XCPUV2'), as found in the S12XE family, features the ability to run in a protected 'user state' (as opposed to supervisor state) by setting the new U bit in the CCR.

Important: RTA-OSEK requires the CPU to operate in the supervisor state at all times. Applications must therefore take care never to set the U bit of the CCR.

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	**MC9S12XEP100**
Clock speed (MHz)	8
Code memory	On-chip FLASH
Read-only data memory	On-chip FLASH
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	16	16	16	16	16	16
Maximum number of not suspended tasks	16	16	16	16	16	16
Maximum number of priorities	16	16	16	16	16	16
Number of tasks per priority (for BCC2 and ECC2)	n/a	16	16	n/a	16	16
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	16	16	16
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	255					

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	27	27	27	27	27	27
	ROM	108	108	113	189	189	194
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	45	45	45	45	45	45
	ROM	173	173	178	254	254	259
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	51	51	51	51	51	51
	ROM	198	198	203	279	279	284
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

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	20	20	20	20	20	20
BCC1 Heavyweight task	RAM	2	2	2	2	2	2
	ROM	22	22	22	22	22	22
BCC2 task	RAM	n/a	3	5	n/a	3	5
	ROM	n/a	25	29	n/a	25	29
ECC1, Integer task	RAM	n/a	n/a	n/a	11	11	11
	ROM	n/a	n/a	n/a	32	32	32
ECC1, floating-point task	RAM	n/a	n/a	n/a	12	12	12
	ROM	n/a	n/a	n/a	32	32	32
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	13
	ROM	n/a	n/a	n/a	n/a	n/a	36
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	14
	ROM	n/a	n/a	n/a	n/a	n/a	36
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	28	28	28	28	28	28
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	41	41	41	41	41	41
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
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	10	10	10	10	10	10
Alarm	RAM	9	9	9	9	9	9
	ROM	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	79	79	79	79	79	79
Message	RAM	11	11	11	31	31	31
	ROM	11	11	11	28	28	28
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10

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	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	6	0	6	6
ScheduleTable	RAM	9	9	9	9	9	9
	ROM	95	95	95	95	95	95
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	9	9	9	9	9	9
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	8	8	8	8	8	8
Arrivalpoint (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8
Schedule	RAM	11	11	11	11	11	11
	ROM	20	20	20	20	20	20
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

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	9	9	9	9	9	9
	ROM	27	27	27	27	27	27
BCC1 Heavyweight task	RAM	11	11	11	11	11	11
	ROM	29	29	29	29	29	29
BCC2 task	RAM	n/a	12	14	n/a	12	14
	ROM	n/a	32	36	n/a	32	36
ECC1, Integer task	RAM	n/a	n/a	n/a	20	20	20
	ROM	n/a	n/a	n/a	39	39	39
ECC1, floating-point task	RAM	n/a	n/a	n/a	21	21	21
	ROM	n/a	n/a	n/a	39	39	39
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	22

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	43
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	23
	ROM	n/a	n/a	n/a	n/a	n/a	43
Category 2 ISR	RAM	9	9	9	9	9	9
	ROM	51	51	51	51	51	51
Category 2 ISR, floating-point	RAM	10	10	10	10	10	10
	ROM	59	59	59	59	59	59
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
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	10	10	10	10	10	10
Alarm	RAM	9	9	9	9	9	9
	ROM	30	30	30	30	30	30
Counter	RAM	4	4	4	4	4	4
	ROM	79	79	79	79	79	79
Message	RAM	11	11	11	31	31	31
	ROM	11	11	11	28	28	28
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	6	0	6	6
ScheduleTable	RAM	9	9	9	9	9	9
	ROM	95	95	95	95	95	95
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	9	9	9	9	9	9
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	8	8	8	8	8	8
Arrivalpoint (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8
Schedule	RAM	11	11	11	11	11	11

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	20	20	20	20	20	20
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

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	10	10	10	10	10	10
	ROM	31	31	31	31	31	31
BCC1 Heavyweight task	RAM	12	12	12	12	12	12
	ROM	31	31	31	31	31	31
BCC2 task	RAM	n/a	13	15	n/a	13	15
	ROM	n/a	34	38	n/a	34	38
ECC1, Integer task	RAM	n/a	n/a	n/a	21	21	21
	ROM	n/a	n/a	n/a	41	41	41
ECC1, floating-point task	RAM	n/a	n/a	n/a	22	22	22
	ROM	n/a	n/a	n/a	41	41	41
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	23
	ROM	n/a	n/a	n/a	n/a	n/a	45
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	24
	ROM	n/a	n/a	n/a	n/a	n/a	45
Category 2 ISR	RAM	10	10	10	10	10	10
	ROM	55	55	55	55	55	55
Category 2 ISR, floating-point	RAM	11	11	11	11	11	11
	ROM	63	63	63	63	63	63
Resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Alarm	RAM	9	9	9	9	9	9
	ROM	32	32	32	32	32	32
Counter	RAM	4	4	4	4	4	4
	ROM	81	81	81	81	81	81
Message	RAM	11	11	11	31	31	31
	ROM	13	13	13	30	30	30
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	6	0	6	6
ScheduleTable	RAM	9	9	9	9	9	9
	ROM	95	95	95	95	95	95
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	9	9	9	9	9	9
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	14	14	14	14	14	14
Arrivalpoint (writable)	RAM	14	14	14	14	14	14
	ROM	14	14	14	14	14	14
Schedule	RAM	15	15	15	15	15	15
	ROM	26	26	26	26	26	26
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

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	91	140	183	95	144	203
	NS		77	126	167	81	130	187
	KL	2	52	102	148	56	106	168
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	17	17	17	17	17	17
ChainTask	SWL	1, 8	83	132	174	87	136	196
	SWH	1, 9	109	154	196	113	158	218
	NSL	8	83	132	174	87	136	196
	NSH	9	103	148	190	107	152	212
Schedule			51	51	72	51	51	72
GetTaskID			35	35	35	35	35	35
GetTaskState			79	79	79	100	100	100
EnableAllInterrupts			7	7	7	7	7	7
DisableAllInterrupts			14	14	14	14	14	14
ResumeAllInterrupts			15	15	15	15	15	15
SuspendAllInterrupts			22	22	22	22	22	22
ResumeOSInterrupts			15	15	15	15	15	15
SuspendOSInterrupts			32	32	32	32	32	32
GetResource	Task	7	22	22	26	22	22	26
	Combined	6	62	62	62	62	62	62
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	42	42	42	42	42	42
	Combined	6	86	86	86	86	86	86
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	90	90	190
	NS		n/a	n/a	n/a	76	76	155
	NS1i	10	n/a	n/a	n/a	32	n/a	n/a
	KL	2	n/a	n/a	n/a	52	52	137
	KL1i	2, 10	n/a	n/a	n/a	13	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28

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	23	23	23
WaitEvent	<default>		n/a	n/a	n/a	177	177	347
	fp	11	n/a	n/a	n/a	201	201	398
	1i	10	n/a	n/a	n/a	21	n/a	n/a
GetAlarmBase			35	35	35	35	35	35
GetAlarm			94	94	94	94	94	94
SetRelAlarm			823	823	823	823	823	823
SetAbsAlarm			966	966	966	966	966	966
CancelAlarm			56	56	56	56	56	56
InitCounter			60	60	60	60	60	60
GetCounterValue			94	94	94	94	94	94
GetScheduleTableStatus		34	53	78	78	53	78	78
NextScheduleTable		34	76	201	201	76	201	201
StartScheduleTable		34	99	145	145	99	145	145
StopScheduleTable		34	69	87	87	69	87	87
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	13	13	13
ScheduleTable expiry point	Callback		5	5	5	5	5	5
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		27	27	27	27	27	27
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	32	32	32	32	32	32
Process container	Non-Yielding	33	13	13	13	13	13	13
osek_tick_alarm	<default>		68	68	68	68	68	68
	KL	2	50	50	50	50	50	50
osek_incr_counter			63	63	63	63	63	63
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			88	88	88	88	88	88
ShutdownOS	NoHook	12	16	16	16	16	16	16
	Hook	13	24	24	24	24	24	24
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			14	14	14	14	14	14
StopCOM			9	9	9	9	9	9
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	48	48	48	130	130	130

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	130	130	130	130	130	130
GetMessageResource			21	21	21	21	21	21
ReleaseMessageResource			19	19	19	19	19	19
GetMessageStatus			37	37	37	37	37	37
SendMessage	SW CCCA	1, 14	79	79	79	202	202	202
	SW CCCB	1, 15	179	179	179	202	202	202
	NS CCCA	14	79	79	79	202	202	202
	NS CCCB	15	179	179	179	202	202	202
	KL CCCA	2, 14	50	50	50	180	180	180
	KL CCCB	2, 15	159	159	159	180	180	180
main_dispatch	NoHook	12	82	82	118	82	82	118
	Hook	13	108	108	144	108	108	144
sub_dispatch	B1LF	19	17	17	17	17	17	17
	B1HI	20	57	57	57	57	57	57
	B1HF	21	65	65	65	65	65	65
	B2LI	22	n/a	41	70	n/a	41	70
	B2LF	23	n/a	49	78	n/a	49	78
	B2HI	24	n/a	156	226	n/a	156	226
	B2HF	25	n/a	164	234	n/a	164	234
	E1HI	26	n/a	n/a	n/a	248	248	320
	E1HF	27	n/a	n/a	n/a	256	256	328
	E2HI	28	n/a	n/a	n/a	n/a	n/a	320
	E2HF	29	n/a	n/a	n/a	n/a	n/a	328
ErrorHook support		16	18	18	18	18	18	18
	ServiceID	17	23	23	23	23	23	23
	Parameters	18	53	53	53	53	53	53
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	87	185	242	97	211	285
	NS		73	172	229	83	197	272
	KL	2	48	151	207	58	176	250
ChainTaskset	SWL	1, 8	82	204	258	94	220	292
	SWH	1, 9	118	248	301	130	266	335
	NSL	8	82	204	258	94	220	292
	NSH	9	112	242	295	124	260	329
GetTasksetRef			21	21	21	21	21	21

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
MergeTaskset			32	32	32	32	32	32
AssignTaskset			10	10	10	10	10	10
RemoveTaskset			32	32	32	32	32	32
TestSubTaskset			55	55	55	55	55	55
TestEquivalentTaskset			49	49	49	49	49	49
TickSchedule	SW	1	194	165	165	165	165	165
	NS		174	152	152	152	152	152
	KL	2	155	134	134	134	134	134
AdvanceSchedule	SW	1	194	163	163	163	163	163
	NS		172	135	135	135	135	135
	KL	2	148	116	116	116	116	116
StartSchedule			62	62	62	62	62	62
StopSchedule			39	39	39	39	39	39
GetScheduleStatus			90	90	90	90	90	90
GetScheduleValue			81	81	81	81	81	81
GetScheduleNext			23	23	23	23	23	23
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			42	42	42	42	42	42
SetArrivalpointDelay			30	30	30	30	30	30
GetArrivalpointTasksetRef			17	17	17	17	17	17
GetArrivalpointNext			21	21	21	21	21	21
SetArrivalpointNext			6	6	6	6	6	6
TestArrivalpointWritable			29	29	29	29	29	29
GetExecutionTime			5	5	5	5	5	5
GetLargestExecutionTime			17	17	17	17	17	17
ResetLargestExecutionTime			2	2	2	2	2	2
GetStackOffset			26	26	26	26	26	26

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	94	143	186	98	147	206
	NS		80	129	167	84	133	187
	KL	2	52	102	148	56	106	168
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	17	17	17	17	17	17
ChainTask	SWL	1, 8	83	132	174	87	136	196
	SWH	1, 9	109	154	196	113	158	218
	NSL	8	83	132	174	87	136	196
	NSH	9	103	148	190	107	152	212
Schedule			70	70	91	70	70	91
GetTaskID			35	35	35	35	35	35
GetTaskState			79	79	79	100	100	100
EnableAllInterrupts			7	7	7	7	7	7
DisableAllInterrupts			14	14	14	14	14	14
ResumeAllInterrupts			15	15	15	15	15	15
SuspendAllInterrupts			22	22	22	22	22	22
ResumeOSInterrupts			15	15	15	15	15	15
SuspendOSInterrupts			32	32	32	32	32	32
GetResource	Task	7	22	22	26	22	22	26
	Combined	6	62	62	62	62	62	62
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	61	61	61	61	61	61
	Combined	6	124	124	124	124	124	124
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	90	90	190
	NS		n/a	n/a	n/a	76	76	155
	NS1i	10	n/a	n/a	n/a	32	n/a	n/a
	KL	2	n/a	n/a	n/a	52	52	137
	KL1i	2, 10	n/a	n/a	n/a	13	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	23	23	23
WaitEvent	<default>		n/a	n/a	n/a	218	218	388
	fp	11	n/a	n/a	n/a	242	242	439

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	57	n/a	n/a
GetAlarmBase			35	35	35	35	35	35
GetAlarm			94	94	94	94	94	94
SetRelAlarm			823	823	823	823	823	823
SetAbsAlarm			965	965	965	965	965	965
CancelAlarm			56	56	56	56	56	56
InitCounter			60	60	60	60	60	60
GetCounterValue			94	94	94	94	94	94
GetScheduleTableStatus		34	53	78	78	53	78	78
NextScheduleTable		34	76	201	201	76	201	201
StartScheduleTable		34	99	145	145	99	145	145
StopScheduleTable		34	69	87	87	69	87	87
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	13	13	13
ScheduleTable expiry point	Callback		5	5	5	5	5	5
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		27	27	27	27	27	27
GetISRID		4	32	32	32	32	32	32
Process container	Yielding	32	32	32	32	32	32	32
Process container	Non-Yielding	33	13	13	13	13	13	13
osek_tick_alarm	<default>		72	72	72	72	72	72
	KL	2	50	50	50	50	50	50
osek_incr_counter			63	63	63	63	63	63
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			126	126	126	126	126	126
ShutdownOS	NoHook	12	16	16	16	16	16	16
	Hook	13	24	24	24	24	24	24
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			14	14	14	14	14	14
StopCOM			9	9	9	9	9	9
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	48	48	48	130	130	130
	CCCB	15	130	130	130	130	130	130
GetMessageResource			21	21	21	21	21	21
ReleaseMessageResource			19	19	19	19	19	19

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			37	37	37	37	37	37
SendMessage	SW CCCA	1, 14	79	79	79	202	202	202
	SW CCCB	1, 15	179	179	179	202	202	202
	NS CCCA	14	79	79	79	202	202	202
	NS CCCB	15	179	179	179	202	202	202
	KL CCCA	2, 14	50	50	50	180	180	180
	KL CCCB	2, 15	159	159	159	180	180	180
main_dispatch	NoHook	12	146	146	191	146	146	191
	Hook	13	177	177	222	177	177	222
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	63	63	63	63	63	63
	B1HF	21	73	73	73	73	73	73
	B2LI	22	n/a	31	60	n/a	31	60
	B2LF	23	n/a	39	68	n/a	39	68
	B2HI	24	n/a	150	220	n/a	150	220
	B2HF	25	n/a	158	228	n/a	158	228
	E1HI	26	n/a	n/a	n/a	267	267	339
	E1HF	27	n/a	n/a	n/a	275	275	347
	E2HI	28	n/a	n/a	n/a	n/a	n/a	339
	E2HF	29	n/a	n/a	n/a	n/a	n/a	347
ErrorHook support		16	18	18	18	18	18	18
	ServiceID	17	23	23	23	23	23	23
	Parameters	18	53	53	53	53	53	53
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	46	46	46	46	46	46
Timing_termination		4	69	69	69	69	69	69
ActivateTaskset	SW	1	90	185	242	100	211	285
	NS		76	172	229	86	197	272
	KL	2	48	151	207	58	176	250
ChainTaskset	SWL	1, 8	82	204	258	94	220	292
	SWH	1, 9	118	248	301	130	266	335
	NSL	8	82	204	258	94	220	292
	NSH	9	112	242	295	124	260	329
GetTasksetRef			21	21	21	21	21	21
MergeTaskset			32	32	32	32	32	32
AssignTaskset			10	10	10	10	10	10
RemoveTaskset			32	32	32	32	32	32

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			49	49	49	49	49	49
TickSchedule	SW	1	194	165	165	165	165	165
	NS		174	152	152	152	152	152
	KL	2	155	134	134	134	134	134
AdvanceSchedule	SW	1	196	165	165	165	165	165
	NS		174	135	135	135	135	135
	KL	2	148	116	116	116	116	116
StartSchedule			62	62	62	62	62	62
StopSchedule			39	39	39	39	39	39
GetScheduleStatus			90	90	90	90	90	90
GetScheduleValue			81	81	81	81	81	81
GetScheduleNext			23	23	23	23	23	23
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			42	42	42	42	42	42
SetArrivalpointDelay			30	30	30	30	30	30
GetArrivalpointTasksetRef			17	17	17	17	17	17
GetArrivalpointNext			21	21	21	21	21	21
SetArrivalpointNext			6	6	6	6	6	6
TestArrivalpointWritable			29	29	29	29	29	29
GetExecutionTime			76	76	76	76	76	76
GetLargestExecutionTime			46	46	46	46	46	46
ResetLargestExecutionTime			31	31	31	31	31	31
GetStackOffset			26	26	26	26	26	26

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	184	231	276	188	235	296
	NS		209	257	299	213	261	319
	KL	2	123	175	218	127	179	238
TerminateTask	LExt	3	76	76	76	76	76	76

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	101	101	101	101	101	101
ChainTask	SWL	1, 8	195	247	289	199	251	307
	SWH	1, 9	221	269	312	225	273	332
	NSL	8	245	300	342	251	304	360
	NSH	9	267	320	363	273	324	386
Schedule			162	162	177	162	162	177
GetTaskID			45	45	45	45	45	45
GetTaskState			197	197	197	205	205	205
EnableAllInterrupts			17	17	17	17	17	17
DisableAllInterrupts			24	24	24	24	24	24
ResumeAllInterrupts			51	51	51	51	51	51
SuspendAllInterrupts			32	32	32	32	32	32
ResumeOSInterrupts			51	51	51	51	51	51
SuspendOSInterrupts			42	42	42	42	42	42
GetResource	Task	7	293	293	244	293	293	244
	Combined	6	291	291	291	291	291	291
	CLEx	3	245	245	245	245	245	245
ReleaseResource	Task	7	255	255	255	255	255	255
	Combined	6	293	293	293	293	293	293
	CLEx	3	237	237	237	237	237	237
SetEvent	SW	1	n/a	n/a	n/a	211	211	312
	NS		n/a	n/a	n/a	260	260	340
	NS1i	10	n/a	n/a	n/a	154	n/a	n/a
	KL	2	n/a	n/a	n/a	173	173	248
	KL1i	2, 10	n/a	n/a	n/a	131	n/a	n/a
ClearEvent			n/a	n/a	n/a	92	92	92
GetEvent			n/a	n/a	n/a	133	133	133
WaitEvent	<default>		n/a	n/a	n/a	305	305	473
	fp	11	n/a	n/a	n/a	329	329	524
	1i	10	n/a	n/a	n/a	144	n/a	n/a
GetAlarmBase			128	128	128	128	128	128
GetAlarm			155	155	155	155	155	155
SetRelAlarm			995	995	995	995	995	995
SetAbsAlarm			1097	1097	1097	1097	1097	1097
CancelAlarm			121	121	121	121	121	121
InitCounter			205	205	205	205	205	205
GetCounterValue			188	188	188	188	188	188

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	63	90	90	63	90	90
NextScheduleTable		34	82	204	204	82	204	204
StartScheduleTable		34	108	155	155	108	155	155
StopScheduleTable		34	77	95	95	77	95	95
ScheduleTable expiry point	ActivateTask		8	8	8	8	8	8
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	13	13	13
ScheduleTable expiry point	Callback		5	5	5	5	5	5
ScheduleTable expiry point	Tick counter		8	8	8	8	8	8
ScheduleTable expiry point	Final		27	27	27	27	27	27
GetISRID		4	42	42	42	42	42	42
Process container	Yielding	32	32	32	32	32	32	32
Process container	Non-Yielding	33	13	13	13	13	13	13
osek_tick_alarm	<default>		102	102	102	102	102	102
	KL	2	50	50	50	50	50	50
osek_incr_counter			63	63	63	63	63	63
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			136	136	136	136	136	136
ShutdownOS	NoHook	12	21	21	21	21	21	21
	Hook	13	29	29	29	29	29	29
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			24	24	24	24	24	24
StopCOM			24	24	24	24	24	24
ReadFlag			18	18	18	18	18	18
ResetFlag			19	19	19	19	19	19
ReceiveMessage	CCCA	14	131	131	131	202	202	202
	CCCB	15	202	202	202	202	202	202
GetMessageResource			60	60	60	60	60	60
ReleaseMessageResource			60	60	60	60	60	60
GetMessageStatus			68	68	68	68	68	68
SendMessage	SW CCCA	1, 14	156	156	156	281	281	281
	SW CCCB	1, 15	259	259	259	281	281	281
	NS CCCA	14	156	156	156	281	281	281
	NS CCCB	15	259	259	259	281	281	281
	KL CCCA	2, 14	111	111	111	237	237	237
	KL CCCB	2, 15	213	213	213	237	237	237
main_dispatch	NoHook	12	146	146	191	146	146	191

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	177	177	222	177	177	222
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	63	63	63	63	63	63
	B1HF	21	73	73	73	73	73	73
	B2LI	22	n/a	31	60	n/a	31	60
	B2LF	23	n/a	39	68	n/a	39	68
	B2HI	24	n/a	150	223	n/a	150	223
	B2HF	25	n/a	158	231	n/a	158	231
	E1HI	26	n/a	n/a	n/a	267	267	344
	E1HF	27	n/a	n/a	n/a	275	275	352
	E2HI	28	n/a	n/a	n/a	n/a	n/a	344
	E2HF	29	n/a	n/a	n/a	n/a	n/a	352
ErrorHook support		16	66	66	66	66	66	66
	ServiceID	17	71	71	71	71	71	71
	Parameters	18	102	102	102	102	102	102
validity_checks		3	26	26	26	26	26	26
Timing_dispatch		4	46	46	46	46	46	46
Timing_termination		4	69	69	69	69	69	69
ActivateTaskset	SW	1	205	254	307	215	277	342
	NS		246	294	350	256	317	385
	KL	2	161	210	263	171	233	299
ChainTaskset	SWL	1, 8	246	307	360	255	322	388
	SWH	1, 9	295	363	418	304	379	442
	NSL	8	298	356	412	307	371	440
	NSH	9	341	406	461	350	422	485
GetTasksetRef			112	112	112	112	112	112
MergeTaskset			173	173	173	173	173	173
AssignTaskset			119	119	119	119	119	119
RemoveTaskset			171	171	171	171	171	171
TestSubTaskset			193	193	193	193	193	193
TestEquivalentTaskset			191	191	191	191	191	191
TickSchedule	SW	1	289	254	254	254	254	254
	NS		341	322	322	322	322	322
	KL	2	242	210	210	210	210	210
AdvanceSchedule	SW	1	346	297	297	297	297	297
	NS		406	379	379	379	379	379
	KL	2	277	236	236	236	236	236

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			186	186	186	186	186	186
StopSchedule			128	128	128	128	128	128
GetScheduleStatus			175	175	175	175	175	175
GetScheduleValue			179	179	179	179	179	179
GetScheduleNext			76	76	76	76	76	76
SetScheduleNext			113	113	113	113	113	113
GetArrivalpointDelay			147	147	147	147	147	147
SetArrivalpointDelay			157	157	157	157	157	157
GetArrivalpointTasksetRef			96	96	96	96	96	96
GetArrivalpointNext			102	102	102	102	102	102
SetArrivalpointNext			127	127	127	127	127	127
TestArrivalpointWritable			108	108	108	108	108	108
GetExecutionTime			108	108	108	108	108	108
GetLargestExecutionTime			146	146	146	146	146	146
ResetLargestExecutionTime			132	132	132	132	132	132
GetStackOffset			26	26	26	26	26	26

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 HC12X16/Metrowerks 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	122	158	228	124	160	237
	NS	113	149	214	115	151	223
	KL	57	93	160	59	93	167
TerminateTask	LExt	0	0	0	0	0	0
	H	150	148	155	148	148	155
ChainTask	SWL	255	296	411	295	334	450
	SWH	315	349	463	354	386	503
	NSL	255	296	411	295	334	450
	NSH	310	344	458	349	381	498
Schedule	SW	136	134	107	134	134	107
GetTaskID		86	85	85	85	85	85
GetTaskState		131	130	131	142	143	142
EnableAllInterrupts		29	29	29	29	29	29
DisableAllInterrupts		47	86	86	47	86	86
ResumeAllInterrupts		37	37	37	37	37	37
SuspendAllInterrupts		55	55	55	55	55	55
ResumeOSInterrupts		37	37	37	37	37	37
SuspendOSInterrupts		55	55	55	55	55	55
GetResource	Task	84	45	85	81	42	82
	Combined	85	85	85	82	82	82
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	87	87	87	85	85	85
	Combined	102	102	102	99	99	99
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	172	172	127
	NS	n/a	n/a	n/a	133	133	125
	KL	n/a	n/a	n/a	77	77	79
ClearEvent		n/a	n/a	n/a	55	55	55
GetEvent		n/a	n/a	n/a	78	78	78
WaitEvent	<default>	n/a	n/a	n/a	353	364	427

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	362	374	436
GetAlarmBase		365	364	325	365	365	364
GetAlarm		148	145	145	148	148	148
SetRelAlarm		191	186	186	191	191	191
SetAbsAlarm		180	175	175	180	180	180
CancelAlarm		97	96	135	97	97	97
InitCounter		114	111	111	114	114	114
GetCounterValue		132	129	129	132	132	132
osek_tick_alarm	<default>	140	139	139	140	140	140
	KL	84	83	83	84	84	84
osek_incr_counter		37	36	36	37	37	37
GetActiveApplicationMode		10	10	10	10	10	10
StartOS		472	471	471	471	471	471
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	56	56	56	56	56	56
InitCOM		57	57	18	60	60	60
CloseCOM		18	18	18	21	21	21
StartCOM		54	54	54	198	198	198
StopCOM		26	26	26	26	26	26
ReadFlag		n/a	n/a	n/a	18	18	18
ResetFlag		n/a	n/a	n/a	10	10	10
ReceiveMessage		87	89	128	234	234	234
GetMessageResource		n/a	n/a	n/a	93	93	93
ReleaseMessageResource		n/a	n/a	n/a	131	131	131
GetMessageStatus		n/a	n/a	n/a	56	56	56
SendMessage	SW	244	282	352	413	449	526
	NS	232	270	335	401	437	509
	KL	124	162	229	297	331	405
ActivateTaskset	SW	133	385	426	139	501	440
	NS	82	333	413	88	449	427
	KL	33	289	368	39	405	382
	SW2	94	346	426	100	462	440
	NS2	82	333	413	88	449	427
	KL2	33	289	368	39	405	382
ChainTaskset	SWL	237	569	737	274	610	777
	SWH	300	629	758	336	673	798

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NSL	237	569	698	274	610	738
	NSH	295	624	753	331	668	793
GetTasksetRef		54	53	54	53	54	53
MergeTaskset		89	89	89	89	89	89
AssignTaskset		33	33	33	33	33	33
RemoveTaskset		88	88	88	88	88	88
TestSubTaskset		114	114	114	114	114	114
TestEquivalentTaskset		107	107	107	107	107	107
TickSchedule	SW	233	518	590	305	683	626
	NS	215	506	578	254	632	614
	KL	164	455	527	203	581	563
	SW2	233	517	596	266	633	610
	NS2	215	505	584	254	621	598
	KL2	164	454	533	203	570	547
AdvanceSchedule	SW	238	531	603	279	657	639
	NS	184	472	544	220	598	580
	KL	130	423	495	171	549	531
	SW2	199	491	570	240	607	584
	NS2	184	471	550	220	587	564
	KL2	130	422	501	171	538	515
StartSchedule		146	146	146	146	146	146
StopSchedule		100	100	100	100	100	100
GetScheduleStatus		129	129	129	129	129	129
GetScheduleValue		125	125	125	125	125	125
GetScheduleNext		52	52	52	52	52	52
SetScheduleNext		38	38	38	38	38	38
GetArrivalpointDelay		105	105	105	105	105	105
SetArrivalpointDelay		96	96	96	96	96	96
GetArrivalpointTasksetRef		44	44	44	44	44	44
GetArrivalpointNext		48	48	48	48	48	48
SetArrivalpointNext		35	35	35	35	35	35
TestArrivalpointWritable		50	50	50	50	50	50
GetExecutionTime		56	56	17	17	17	17
GetLargestExecutionTime		70	69	69	69	69	69
ResetLargestExecutionTime		19	18	18	18	18	18
GetStackOffset		68	68	68	68	68	68

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	125	162	229	127	165	238
	NS	116	153	214	118	156	223
	KL	59	96	160	61	97	167
TerminateTask	LExt	0	0	0	0	0	0
	H	546	541	547	541	541	547
ChainTask	SWL	696	734	859	735	777	899
	SWH	789	781	905	827	823	946
	NSL	696	734	859	735	777	899
	NSH	743	774	898	781	816	939
Schedule	SW	99	97	110	97	97	110
GetTaskID		47	46	85	46	46	85
GetTaskState		132	131	131	144	145	142
EnableAllInterrupts		29	29	29	29	29	29
DisableAllInterrupts		86	86	47	86	86	47
ResumeAllInterrupts		37	37	37	37	37	37
SuspendAllInterrupts		55	55	55	55	55	55
ResumeOSInterrupts		37	37	37	37	37	37
SuspendOSInterrupts		55	55	55	55	55	55
GetResource	Task	46	46	85	43	43	82
	Combined	85	85	85	82	82	82
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	87	87	87	85	85	85
	Combined	102	102	102	99	99	99
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	173	173	167
	NS	n/a	n/a	n/a	134	134	126
	KL	n/a	n/a	n/a	78	78	80
ClearEvent		n/a	n/a	n/a	55	55	55
GetEvent		n/a	n/a	n/a	78	78	78
WaitEvent	<default>	n/a	n/a	n/a	732	746	804
	fp	n/a	n/a	n/a	741	756	813
GetAlarmBase		326	324	324	325	326	325
GetAlarm		148	145	145	148	148	148

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		191	186	186	191	191	191
SetAbsAlarm		180	175	175	180	180	180
CancelAlarm		97	96	96	97	97	97
InitCounter		114	111	111	114	114	114
GetCounterValue		132	129	129	132	132	132
osek_tick_alarm	<default>	145	144	144	145	145	145
	KL	84	83	83	84	84	84
osek_incr_counter		37	36	36	37	37	37
GetActiveApplicationMode		10	10	10	10	10	10
StartOS		1111	1110	1110	1109	1109	1109
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	56	56	56	56	56	56
InitCOM		18	18	18	21	21	21
CloseCOM		18	18	18	21	21	21
StartCOM		54	54	54	198	198	198
StopCOM		26	26	26	26	26	26
ReadFlag		n/a	n/a	n/a	18	18	18
ResetFlag		n/a	n/a	n/a	10	10	10
ReceiveMessage		87	89	89	234	234	234
GetMessageResource		n/a	n/a	n/a	94	94	94
ReleaseMessageResource		n/a	n/a	n/a	131	131	131
GetMessageStatus		n/a	n/a	n/a	56	56	56
SendMessage	SW	247	286	353	416	454	527
	NS	235	274	335	404	442	509
	KL	126	165	229	299	335	405
ActivateTaskset	SW	95	346	426	101	462	440
	NS	83	333	413	89	449	427
	KL	33	289	368	39	405	382
	SW2	95	346	426	101	462	440
	NS2	83	333	413	89	449	427
	KL2	33	289	368	39	405	382
ChainTaskset	SWL	675	1005	1147	711	1050	1188
	SWH	731	1058	1200	766	1106	1280
	NSL	675	1005	1147	711	1050	1188
	NSH	726	1051	1193	761	1099	1234
GetTasksetRef		54	53	54	53	54	53

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		89	89	89	89	89	89
AssignTaskset		33	33	33	33	33	33
RemoveTaskset		88	88	88	88	88	88
TestSubTaskset		114	114	114	114	114	114
TestEquivalentTaskset		107	107	107	107	107	107
TickSchedule	SW	233	518	590	266	644	626
	NS	215	506	578	254	632	614
	KL	164	455	527	203	581	563
	SW2	233	517	596	266	633	610
	NS2	215	505	584	254	621	598
	KL2	164	454	533	203	570	547
AdvanceSchedule	SW	200	493	565	241	619	601
	NS	185	472	544	220	598	580
	KL	130	423	495	171	549	531
	SW2	200	492	571	241	608	585
	NS2	185	471	550	220	587	564
	KL2	130	422	501	171	538	515
StartSchedule		146	126	146	146	146	146
StopSchedule		100	100	100	100	100	100
GetScheduleStatus		129	129	129	129	129	129
GetScheduleValue		125	125	125	125	125	125
GetScheduleNext		52	52	52	52	52	52
SetScheduleNext		38	38	38	38	38	38
GetArrivalpointDelay		105	105	105	105	105	105
SetArrivalpointDelay		96	96	96	96	96	96
GetArrivalpointTasksetRef		44	44	44	44	44	44
GetArrivalpointNext		48	48	48	48	48	48
SetArrivalpointNext		35	35	35	35	35	35
TestArrivalpointWritable		50	50	50	50	50	50
GetExecutionTime		225	225	264	224	224	224
GetLargestExecutionTime		139	137	137	137	137	137
ResetLargestExecutionTime		83	81	81	81	81	81
GetStackOffset		68	68	68	68	68	68

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	394	411	478	370	414	487
	NS	419	433	495	393	437	503
	KL	316	371	433	292	374	403
TerminateTask	LExt	593	587	593	587	587	593
	H	635	628	634	628	628	634
ChainTask	SWL	1033	1046	1171	1044	1094	1209
	SWH	1082	1090	1214	1092	1132	1250
	NSL	1078	1091	1216	1089	1140	1253
	NSH	1120	1128	1252	1130	1171	1292
Schedule	SW	149	146	157	146	146	157
GetTaskID		94	93	93	93	93	93
GetTaskState		412	385	385	391	392	389
EnableAllInterrupts		37	37	37	37	37	37
DisableAllInterrupts		55	55	55	55	55	55
ResumeAllInterrupts		51	51	51	51	51	51
SuspendAllInterrupts		63	63	63	63	63	63
ResumeOSInterrupts		51	51	51	51	51	51
SuspendOSInterrupts		63	63	63	63	63	63
GetResource	Task	828	795	389	797	800	407
	Combined	335	335	335	356	356	356
	CLEx	363	359	359	382	382	382
ReleaseResource	Task	328	328	328	349	349	349
	Combined	301	301	301	323	323	323
	CLEx	329	326	326	350	350	350
SetEvent	SW	n/a	n/a	n/a	416	416	416
	NS	n/a	n/a	n/a	410	410	410
	KL	n/a	n/a	n/a	339	339	328
ClearEvent		n/a	n/a	n/a	85	85	85
GetEvent		n/a	n/a	n/a	321	321	321
WaitEvent	<default>	n/a	n/a	n/a	810	825	878
	fp	n/a	n/a	n/a	819	835	887
GetAlarmBase		492	478	491	493	492	507
GetAlarm		317	301	314	317	317	332

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		446	420	433	446	446	461
SetAbsAlarm		397	374	387	397	397	412
CancelAlarm		271	257	270	271	271	286
InitCounter		468	438	459	468	468	491
GetCounterValue		289	275	275	289	289	289
osek_tick_alarm	<default>	180	179	179	180	180	180
	KL	84	83	83	84	84	84
osek_incr_counter		37	36	36	37	37	37
GetActiveApplicationMode		10	10	10	10	10	10
StartOS		1135	1134	1134	1133	1133	1133
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	60	60	60	60	60	60
InitCOM		18	18	18	21	21	60
CloseCOM		18	18	18	21	21	21
StartCOM		66	66	66	210	210	210
StopCOM		37	37	37	37	37	37
ReadFlag		n/a	n/a	n/a	41	41	41
ResetFlag		n/a	n/a	n/a	34	34	34
ReceiveMessage		225	234	234	362	362	362
GetMessageResource		n/a	n/a	n/a	535	535	535
ReleaseMessageResource		n/a	n/a	n/a	469	469	469
GetMessageStatus		n/a	n/a	n/a	150	150	150
SendMessage	SW	644	670	737	789	833	906
	NS	666	689	751	809	853	919
	KL	508	533	595	648	691	759
ActivateTaskset	SW	401	752	758	408	749	763
	NS	429	779	910	435	776	926
	KL	340	691	697	347	688	702
	SW2	401	752	758	408	749	763
	NS2	429	779	910	435	776	926
	KL2	340	691	697	347	688	702
ChainTaskset	SWL	1085	1443	1512	1126	1479	1544
	SWH	1315	1626	1769	1316	1667	1808
	NSL	1125	1482	1676	1165	1518	1719
	NSH	1311	1659	1802	1350	1700	1841
GetTasksetRef		303	276	277	276	277	276

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		163	163	163	163	163	163
AssignTaskset		83	83	83	83	83	83
RemoveTaskset		159	159	159	159	159	159
TestSubTaskset		191	191	191	191	191	191
TestEquivalentTaskset		185	185	185	185	185	185
TickSchedule	SW	303	972	969	625	998	1021
	NS	352	1005	1002	658	1031	1054
	KL	227	898	895	551	924	947
	SW2	303	970	976	625	967	981
	NS2	352	1003	1009	658	1000	1014
	KL2	227	896	902	551	893	907
AdvanceSchedule	SW	284	949	946	602	975	998
	NS	356	995	992	648	1021	1044
	KL	206	873	870	526	899	922
	SW2	284	947	953	602	944	958
	NS2	356	993	999	648	990	1004
	KL2	206	871	877	526	868	882
StartSchedule		225	225	225	225	225	206
StopSchedule		152	152	152	152	152	169
GetScheduleStatus		174	174	174	174	174	174
GetScheduleValue		165	165	165	165	165	165
GetScheduleNext		68	68	68	68	68	68
SetScheduleNext		87	87	87	87	87	87
GetArrivalpointDelay		158	158	158	158	158	158
SetArrivalpointDelay		172	172	172	172	172	172
GetArrivalpointTasksetRef		64	64	64	64	64	64
GetArrivalpointNext		69	69	69	69	69	69
SetArrivalpointNext		93	93	93	93	93	93
TestArrivalpointWritable		70	70	70	70	70	70
GetExecutionTime		296	296	296	295	295	256
GetLargestExecutionTime		401	374	374	374	374	374
ResetLargestExecutionTime		367	340	340	340	340	340
GetStackOffset		68	68	68	68	68	68

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	38	38	38	38	38	38
	Cat 2	50	125	125	125	125	125

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	38	38	38	38	38	38
	Cat 2	376	444	444	443	443	443

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	38	38	38	38	38	38
	Cat 2	376	444	444	443	443	443

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.

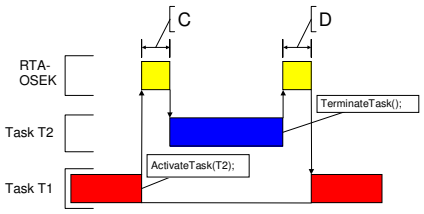

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

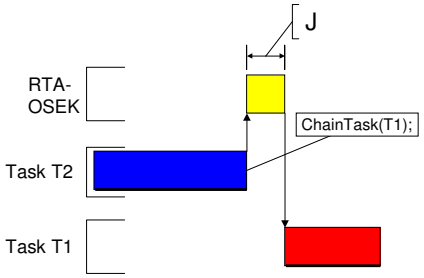

Figure 2: Task Chaining

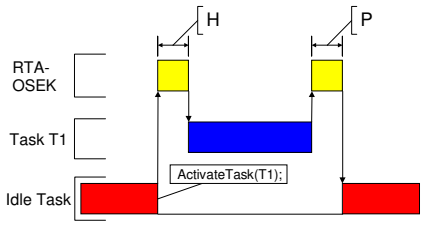

Figure 3: Task Activation from Idle Task

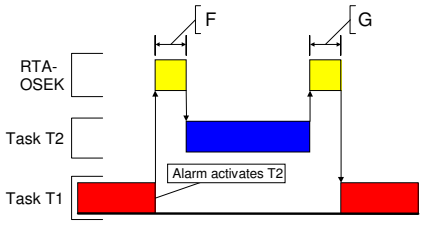

Figure 4: Task Activation from an Alarm

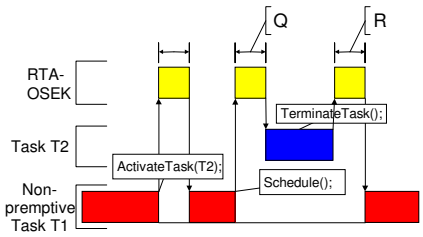

Figure 5: Non-Premptive Task Calls Schedule()

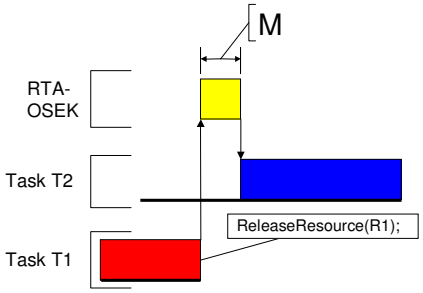

Figure 6: Blocked Task Activated by ReleaseResource()

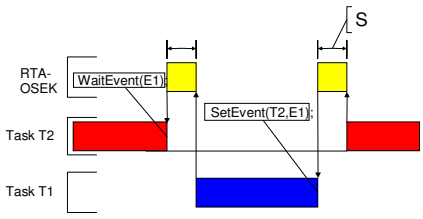

Figure 7: Waiting Task Activated by SetEvent()

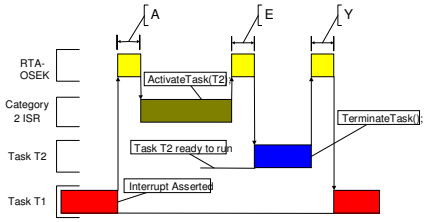

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	87	125	162	86	125	167
Figure 1: D	Heavy, Basic/Extended	150	178	222	188	192	226
ChainTask	Light, Basic	199	254	363	196	248	368
Figure 2: J	Heavy, Basic/Extended	523	598	751	557	607	761
Pre-emption	Light, Basic	183	236	343	182	236	378
Figure 1: C	Heavy, Basic/Extended	239	277	404	280	321	446
From idle task	Light, Basic	183	236	343	182	236	378
Figure 3: H	Heavy, Basic/Extended	239	277	404	280	321	446
Triggered by alarm	Light, Basic	347	399	506	345	399	541
Figure 4: F	Heavy, Basic/Extended	403	440	567	443	484	609
Schedule	Light, Basic	149	160	223	145	161	229
Figure 5: Q	Heavy, Basic/Extended	205	206	274	243	249	308
Release resource	Light, Basic	159	171	223	153	168	225
Figure 6: M	Heavy, Basic/Extended	215	217	274	251	256	304
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	414	427	614
From category 2 ISR	Light, Basic	118	232	284	217	232	289
Figure 8: E	Heavy, Basic/Extended	174	278	335	315	320	368

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	490	513	545	486	513	553
Figure 1: D	Heavy, Basic/Extended	545	555	602	566	573	603
ChainTask	Light, Basic	646	693	814	640	684	816
Figure 2: J	Heavy, Basic/Extended	1361	1410	1578	1373	1422	1584
Pre-emption	Light, Basic	497	544	671	494	542	706
Figure 1: C	Heavy, Basic/Extended	549	587	730	590	634	771
From idle task	Light, Basic	497	544	671	494	542	706
Figure 3: H	Heavy, Basic/Extended	549	587	730	590	634	771
Triggered by alarm	Light, Basic	666	712	839	662	710	874
Figure 4: F	Heavy, Basic/Extended	718	755	898	758	802	939
Schedule	Light, Basic	465	468	548	459	471	557
Figure 5: Q	Heavy, Basic/Extended	515	513	600	553	562	637
Release resource	Light, Basic	472	478	547	464	473	548
Figure 6: M	Heavy, Basic/Extended	522	523	599	558	564	628
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	701	714	915
From category 2 ISR	Light, Basic	843	941	1010	930	939	1014
Figure 8: E	Heavy, Basic/Extended	893	986	1062	1024	1030	1094

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	594	616	647	588	616	656
Figure 1: D	Heavy, Basic/Extended	634	642	690	653	661	690
ChainTask	Light, Basic	983	1005	1126	949	996	1126
Figure 2: J	Heavy, Basic/Extended	1782	1806	1975	1764	1819	1981
Pre-emption	Light, Basic	762	789	916	733	787	951
Figure 1: C	Heavy, Basic/Extended	814	832	975	829	879	1016
From idle task	Light, Basic	762	789	916	733	787	951
Figure 3: H	Heavy, Basic/Extended	814	832	975	829	879	1016
Triggered by alarm	Light, Basic	966	992	1119	936	990	1154
Figure 4: F	Heavy, Basic/Extended	1018	1035	1178	1032	1082	1219
Schedule	Light, Basic	510	512	591	503	516	601
Figure 5: Q	Heavy, Basic/Extended	560	557	643	597	607	681
Release resource	Light, Basic	683	689	758	699	708	783
Figure 6: M	Heavy, Basic/Extended	733	734	810	793	799	863
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	944	957	1161
From category 2 ISR	Light, Basic	875	973	1042	962	971	1046
Figure 8: E	Heavy, Basic/Extended	925	1018	1094	1056	1062	1126

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	22	22	25	22	22	25
BCC1 lightweight, floating-point	25	25	28	25	25	28
BCC1 heavyweight, integer	30	30	33	30	30	33
BCC1 heavyweight, floating-point	30	30	33	30	30	33
BCC2 lightweight, integer	n/a	25	30	n/a	25	30
BCC2 lightweight, floating-point	n/a	25	30	n/a	25	30
BCC2 heavyweight, integer	n/a	34	41	n/a	34	41
BCC2 heavyweight, floating-point	n/a	34	41	n/a	34	41
ECC1 heavyweight, integer	n/a	n/a	n/a	40	40	39
ECC1 heavyweight, floating-point	n/a	n/a	n/a	40	40	39
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	43
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	43
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	25	25	25	25	25	25
BCC1 lightweight, floating-point	28	28	28	28	28	28
BCC1 heavyweight, integer	33	33	33	33	33	33
BCC1 heavyweight, floating-point	33	33	33	33	33	33
BCC2 lightweight, integer	n/a	28	30	n/a	28	30
BCC2 lightweight, floating-point	n/a	28	30	n/a	28	30
BCC2 heavyweight, integer	n/a	37	41	n/a	37	41
BCC2 heavyweight, floating-point	n/a	37	41	n/a	37	41
ECC1 heavyweight, integer	n/a	n/a	n/a	39	39	39
ECC1 heavyweight, floating-point	n/a	n/a	n/a	39	39	39
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	43
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	43

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	31	31	34	31	31	34
BCC1 lightweight, floating-point	34	34	37	34	34	37
BCC1 heavyweight, integer	39	39	42	39	39	42
BCC1 heavyweight, floating-point	39	39	42	39	39	42
BCC2 lightweight, integer	n/a	34	39	n/a	34	39
BCC2 lightweight, floating-point	n/a	34	39	n/a	34	39
BCC2 heavyweight, integer	n/a	43	50	n/a	43	50
BCC2 heavyweight, floating-point	n/a	43	50	n/a	43	50
ECC1 heavyweight, integer	n/a	n/a	n/a	49	49	48
ECC1 heavyweight, floating-point	n/a	n/a	n/a	49	49	48
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	52
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	52
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	34	34	34	34	34	34
BCC1 lightweight, floating-point	37	37	37	37	37	37
BCC1 heavyweight, integer	42	42	42	42	42	42
BCC1 heavyweight, floating-point	42	42	42	42	42	42
BCC2 lightweight, integer	n/a	37	39	n/a	37	39
BCC2 lightweight, floating-point	n/a	37	39	n/a	37	39
BCC2 heavyweight, integer	n/a	46	50	n/a	46	50
BCC2 heavyweight, floating-point	n/a	46	50	n/a	46	50
ECC1 heavyweight, integer	n/a	n/a	n/a	48	48	48
ECC1 heavyweight, floating-point	n/a	n/a	n/a	48	48	48
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	52
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	52

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	31	31	34	31	31	34
BCC1 lightweight, floating-point	34	34	37	34	34	37
BCC1 heavyweight, integer	39	39	42	39	39	42
BCC1 heavyweight, floating-point	39	39	42	39	39	42
BCC2 lightweight, integer	n/a	34	39	n/a	34	39
BCC2 lightweight, floating-point	n/a	34	39	n/a	34	39
BCC2 heavyweight, integer	n/a	43	50	n/a	43	50
BCC2 heavyweight, floating-point	n/a	43	50	n/a	43	50
ECC1 heavyweight, integer	n/a	n/a	n/a	49	49	48
ECC1 heavyweight, floating-point	n/a	n/a	n/a	49	49	48
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	52
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	52
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	34	34	34	34	34	34
BCC1 lightweight, floating-point	37	37	37	37	37	37
BCC1 heavyweight, integer	42	42	42	42	42	42
BCC1 heavyweight, floating-point	42	42	42	42	42	42
BCC2 lightweight, integer	n/a	37	39	n/a	37	39
BCC2 lightweight, floating-point	n/a	37	39	n/a	37	39
BCC2 heavyweight, integer	n/a	46	50	n/a	46	50
BCC2 heavyweight, floating-point	n/a	46	50	n/a	46	50
ECC1 heavyweight, integer	n/a	n/a	n/a	48	48	48
ECC1 heavyweight, floating-point	n/a	n/a	n/a	48	48	48
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	52
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	52

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for the HC12X16/Metrowerks 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 and a reduced Context Save Area usage. 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 Version 5.0.5

6.1 Variants

Support for the S12XF, S12XHZ and S12XS families of devices has been added.

6.2 Compiler

The kernel is now built and tested with CodeWarrior v5.0.

6.3 Memory model

The kernel is build for use with the **large** memory model, as opposed to the banked model used previously.

Users migrating applications from the previous banked memory model versions should pay particular attention to:

- Compiler flags
- Linker scripts and placement of data
- Access to global data through the page registers
- Changes in code size and speed

6.4 Floating Point Wrappers

The default implementation of the floating point wrappers preserves the EPAGE register.

7 Version 5.0.2

7.1 Compiler

The kernel is now built and tested with CodeWarrior v4.6.3 instead of v4.5.

8 Version 5.0.1

8.1 Variants

Support for the S12XE family of devices has been added.

8.2 Compiler

The kernel is now built and tested with CodeWarrior v4.5 instead of v4.1.

8.3 ORTI Support

ORTI support for the iSYSTEM winIDEA and Lauterbach TRACE32 debuggers has been added.

9 Version 5.0.0

9.1 Floating Point Wrappers

The default implementation of the floating point wrappers preserves the GPAGE register.

10 Compatibility with Pre-v5 Kernels

10.1 Updating the Application Version

To convert an existing v3.x OIL configuration file to v5.00, load the file into the v5 RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.00. When the OIL configuration file is saved it will then use the v5.0 format and the v5.00 kernel libraries. This process can be reversed to move back to earlier kernel versions.

10.2 32 Bit Timer Drivers

The v3.x kernel uses sixteen bit timer values, whereas the v5.00 kernel uses thirty-two bit timer values. Therefore any existing applications' timer drivers will need modifying. Since the S12X Timer Module provides only sixteen bit timer registers the upper sixteen bits will need to be emulated in software. The provided example application demonstrates one method of achieving this for the `TCNT` timer register.

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.