RTA-OSEK

Binding Manual: V850E/GreenHills

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

Tel.:+49 (711) 8 96 61-102
Fax:+49 (711) 8 96 61-106

www.etas.de

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

Tel.: +1 (888) ETAS INC
Fax: +1 (734) 997-94 49

www.etasinc.com

Japan

ETAS K.K.
Queen's Tower C-17F,
2-3-5, Minatomirai, Nishi-ku,
Yokohama, Kanagawa
220-6217 Japan

Tel.: +81 (45) 222-0900
Fax: +81 (45) 222-0956

www.etas.co.jp

France

ETAS S.A.S.
1, place des États-Unis
SILIC 307
94588 Rungis Cedex

Tel.: +33 (1) 56 70 00 50
Fax: +33 (1) 56 70 00 51

www.etas.fr

Korea

ETAS Korea Co. Ltd.
4F, 705 Bldg. 70-5
Yangjae-dong, Seocho-gu
Seoul 137-889, Korea

Tel.: +82 (2) 57 47-016
Fax: +82 (2) 57 47-120

www.etas.co.kr

Great Britain

ETAS UK Ltd.
Studio 3, Waterside Court
Third Avenue, Centrum 100
Burton-upon-Trent
Staffordshire DE14 2WQ

Tel.: +44 (0) 1283 - 54 65 12
Fax: +44 (0) 1283 - 54 87 67

www.etas-uk.net

Copyright Notice

Version: M00080-001

Disclaimer

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

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

Trademarks

RTA-OSEK, 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 5
1.1 Who Should Read this Guide? 5
1.2 Conventions 5
2 Toolchain Issues 7
2.1 Compiler 7
2.2 Assembler 8
2.3 Linker/Locator 8
2.4 Debugger 9
3 Target Hardware Issues 11
3.1 Interrupts 11
3.1.1 Interrupt Levels 11
3.1.2 Interrupt Vectors 11
3.1.3 Category 1 Handlers 11
3.1.4 Category 2 Handlers 12

3.1.5 Vector Table Issues 12
3.1.6 Traps 14
3.1.7 Default Interrupt 15
3.1.8 Flash Security ID 15
3.2 Register Settings 16
3.3 Stack Usage 17
3.3.1 Number of Stacks 17
3.3.2 Stack Usage within API Calls 17
4 Parameters of Implementation 19
4.1 Functionality 19
4.2 Hardware Resources 20
4.2.1 ROM and RAM Overheads 20
4.2.2 ROM and RAM for OSEK OS Objects 21
4.2.3 Size of Linkable Modules 26
4.2.4 Reserved Hardware Resources 40
4.3 Performance 40
4.3.1 Execution Times for RTA-OSEK API Calls 40
4.3.2 OS Start-up Time 50
4.3.3 Interrupt Latencies 50
4.3.4 Task Switching Times 51
4.4 Configuration of Run-time Context 54
5 Compatibility with Pre-v5 Kernels 58
5.1 Updating the Application Version 58
5.2 32 Bit Timer Drivers 58
5.3 Vector Table 58

1 About this Guide

This guide provides target-specific information for the V850E/GreenHills 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	Green Hills Software, Inc.
Compiler	C-V850E
Version	4.2.1 (MULTI 4.2.3 patch 1)

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

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

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

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

Option	Description
`-O`	Use optimizations
`-noobj`	Turn off binary code generation
`-r21has65535`	r20 always contains 255, r21 always contains 65535
`-registermode=26`	Do not use r17-r22
`-g`	Turn debug information on
`-dual_debug`	Generate 'native' (e.g. DWARF) debugging information in addition to the Green Hills .dbo format
`-dwarf`	Generate DWARF debugging information

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

Option	Description
-zda	Use ZDA
-sda	Use SDA

2.2 Assembler

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

Vendor	Green Hills Software, Inc.
Assembler	AS-V850
Version	4.0 (MULTI 4.2.3 patch 1)

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

2.3 Linker/Locator

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

Sections	ROM/RAM	Description
os_pid	ROM	RTA-OSEK read-only data.
os_pird	ROM	RTA-OSEK initialization data. This is only accessed during StartOS().
os_intvec	ROM	Vector table (if generated by RTA-OSEK).
os_wrappers	ROM	RTA-OSEK interrupt wrappers.
os_text	ROM	RTA-OSEK code section.
os_pir	RAM	RTA-OSEK initialized data. Initialized from os_pird during StartOS().
os_pur	RAM	RTA-OSEK uninitialized data. Must be zeroed during C-startup.
os_pirf	RAM	RTA-OSEK initialized data. Must be initialized during C-startup.
os_trace_ram	RAM	RTA-TRACE uninitialized data. Must be zeroed during C-startup.

In order to avoid the cost of the C-startup unnecessarily clearing `os_pir`, this section should be given the `NOCLEAR` attribute in the linker directive (`*.ld`) file. The supplied example application's linker directive file shows how this can be performed.

2.4 Debugger

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

ORTI compatible debuggers	iSYSTEM winIDEA

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

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

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for V850E/GreenHills. You can find out more about configuring interrupts for RTA-OSEK in the *RTA-OSEK User Guide*.

3.1.1 Interrupt Levels

In RTA-OSEK interrupts are allocated an Interrupt Priority Level (IPL). This is a processor independent abstraction of the interrupt priorities that are available on the target hardware. You can find out more about IPLs in the *RTA-OSEK User Guide*. The hardware interrupt controller is explained in the *appropriate NEC manuals*.

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

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

3.1.2 Interrupt Vectors

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

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

The valid base addresses for the vector table are:

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

3.1.3 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Green

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

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

The number of vectors available depends upon the specific V850 chip variant selected in RTA-OSEK. The variants directly supported are DJ3, FE2, FF2, FF3, FG2, FG3, FJ2, FJ3, FK3, PH2, PH3, RS1, SG2, SG3, SJ2 and SJ3. These are in addition to the 'Generic V850E' variant. Further variants can be supported by contacting LiveDevices.

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

RTA-OSEK generates three different interrupt-handling assembler source files, each with a different approach to supporting ISRs. They are mutually exclusive: only one of the three should be compiled and linked into an application. `osvec1.850` is the only one to include a vector table; `osvec2.850` and `osvec3.850` both require a vector table to be supplied by the user. As such, it is recommended that `osvec1.850` is used unless the added flexibility of `osvec2.850` or `osvec3.850` is required. An explanation of the contents of each of the files follows below.

Note that when you configure your application with the RTA-OSEK GUI you can choose whether or not a vector table is generated. This option dictates whether or not the file `osvec1.850` is generated; `osvec2.850` and `osvec3.850` are always generated, regardless of this option.

In the following discussion, an 'outer wrapper' is a small function, specific to an ISR, which sets up sufficient context for that ISR's entry function pointer to be passed along to the 'mid-wrapper.' The mid-wrapper is common to all

Category 2 ISRs and saves and restores the register context around the call to the ISR's entry function.

osvec1.850

The file `osvec1.850` contains the interrupt vector table (containing the outer wrappers) and the mid-wrapper. The vector table is placed in the `os_intvec` section, which should be linked starting at address 0x10, and the mid-wrapper is placed in the `os_wrappers` section.

osvec2.850

The file `osvec2.850` does not contain a traditional vector table, but does contain a 'jump table' with the label `_os_vec2_table`, which is placed in the `os_jumptable` section. The table contains four-byte entries, one for each vector from `0x10` up to the highest bound vector. The content of each table entry depends on its corresponding vector as follows:

- If a Category 2 ISR is bound to the vector, the entry is a jump (`jr`) to the outer interrupt wrapper for that ISR.
- If a Category 1 ISR is bound to the vector, the entry is a jump to the ISR's entry function.
- If the vector is unbound and there exists a default interrupt, the entry is a jump to the default interrupt's entry function.
- If the vector is unbound and there is no default interrupt, the entry is a `nop` instruction.

The file also contains each outer wrapper referenced in the jump table, and the mid-wrapper. These are all placed in the `os_wrappers` section.

The file can be assembled with only the wrappers (i.e. without the jump table) by defining the symbol `OS_NO_JUMP_TABLE` on the command line.

osvec3.850

The file `osvec3.850` contains a jump table similar to that in `osvec2.850`, with the difference that the entries contain addresses rather than jump instructions. The content of each four-byte table entry depends on its corresponding vector as follows:

- If a Category 2 ISR is bound to the vector, the entry is the address of the outer interrupt wrapper for that ISR.
- If a Category 1 ISR is bound to the vector, the entry is the address of the ISR's entry function.
- If the vector is unbound and there exists a default interrupt, the entry is the address of the default interrupt's entry function.
- If the vector is unbound and there is no default interrupt, the entry is zero.

The file also contains the outer wrappers referenced in the jump table, which differ from the wrappers in `osvec2.850` by omitting the code to preserve `r6` on the stack.

`osvec3.850` also contains a special form of mid-wrapper. Unlike the 'regular' mid-wrapper used in `osvec1.850` and `osvec2.850`, this mid-wrapper does not restore `r6` from the stack after the ISR has run, and does not return from interrupt (`reti`). The final instruction of the mid-wrapper is a jump to `os_end_wrapper`. A default implementation of `os_end_wrapper`, which restores `r6` from the stack and returns from interrupt, is provided. The default implementation can be removed by the preprocessor by defining the symbol `OS_NO_END_WRAPPER` on the command line.

Important: When using `osvec3.850`, it is the responsibility of the user to preserve `r6` in interrupt-handling code before execution reaches the outer wrapper. The default implementation of `os_end_wrapper` can be used if only `r6` is preserved on the stack. If any other registers are used before execution reaches the outer wrappers, then they must be preserved on the stack, and `os_end_wrapper` overridden to restore them (and `r6`) from the stack. Any such additional stack usage must be accounted for in the idle task's stack usage.

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	Wrapper Label (Osvec[23].850 Only)
0xVVVV	_os_wrapper_vvvv
e.g. 0x03A0	_os_wrapper_03a0

3.1.6 Traps

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

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

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

```
}

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

 DisableAllInterrupts();
 ...
 EnableAllInterrupts();

 set_PSW(psw_val);
}
```

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

3.1.7 Default Interrupt

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

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

3.1.8 Flash Security ID

To protect the contents of internal ROM some V850 variants such as the FE2 and FF2 support a 10 byte security number located at memory address 0x70. This address falls within the interrupt vector table range. If the user wishes to enter a 10 byte security number at this address there are three available methods:

1. The `OS_SECURITY_ID` macro in `osvec1.850` can be defined on the command line. For example, assembling `osvec1.850` with the command line option `-DOS_SECURITY_ID=0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA` will insert the security number 0x112233445566778899AA at address 0x70.
2. `OS_SECURITY_ID` can alternatively be defined in a header file, and `osvec1.850` can be made to include that file by defining `OS_SECURITY_ID_HEADER` to be the file's name. For example, assembling

osvec1.850 with the option -DOS_SECURITY_ID_HEADER =\"security_id.h\" will have the effect of including security_id.h at the start of osvec1.850.

3. The security ID can also be specified in an assembler source file, and osvec1.850 can be made to include that file at vector 0x70. This is achieved using the preprocessor symbol OS_SECURITY_ID_ASM. For example, creating the file security_id.850 containing ".byte 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA" and assembling osvec1.850 with the option –DOS_SECURITY_ID_ASM=\"security_id.850\" will insert the security ID 0x112233445566778899AA at address 0x70.

3.2 Register Settings

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

Register	Required Value	Notes
r20	255	
r21	65535	

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

Register	Notes
PSW.ID	Processor status word interrupt enable bit

The kernel has been compiled with the -registermode=26 option. This means that registers r17 – r22 are free for use by applications without risk of them being corrupted by the kernel. However, r17 – r22 are preserved neither by the interrupt wrappers nor by setjmp/longjmp, so applications which use these registers (e.g., by compiling with the default option -registermode=32) must take care to preserve them in the floating-point wrappers. The provided wrappers preserve all six of these registers and may be freely modified to suit differing register usage.

In order to make use of these wrappers all tasks and ISRs which modify registers r17 – r22 should be identified as 'using floating-point' in the RTA-OSEK GUI, as demonstrated in the example application. Further details on the floating-point wrappers may be found in the RTA-OSEK User Guide.

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): 64

Timing

API max usage (bytes): 64

Extended

API max usage (bytes): 88

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.

4 Parameters of Implementation

This chapter provides detailed information on the functionality, performance and memory demands of the RTA-OSEK Component.

The RTA-OSEK Component is highly scalable. As a result, different figures will be obtained when your application uses different sets of features. These feature-sets give six classes of RTA-OSEK, depending on whether your application uses events, shared task priorities and/or multiple (queued) task activations. You should identify which class your application belongs to and then use the figures from the appropriate column in the table.

The following hardware was used to take the measurements in this chapter:

Processor	**V850E/PH2**
Clock speed (MHz)	64
Code memory	On-chip FLASH ROM
Read-only data memory	On-chip FLASH ROM
Read-write data memory	On-chip RAM

4.1 Functionality

The OSEK Operating System Specification specifies four conformance classes. These attributes apply to *systems* built with OSEK OS objects. The following table specifies the number of OSEK OS and COM objects supported per conformance class.

Configuration	**Application Uses**					
Events	**No**			**Yes**		
Shared Task Priorities	**No**		**Yes**	**No**		**Yes**
Multiple Task Activations	**No**	**Yes**		**No**	**Yes**	
Maximum number of tasks	64	64	64	64	64	64
Maximum number of not suspended tasks	64	64	64	64	64	64
Maximum number of priorities	64	64	64	64	64	64
Number of tasks per priority (for BCC2 and ECC2)	n/a	64	64	n/a	64	64
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	32	32	32
Limits for the number of alarm objects (per system / per task)	not limited by RTA-OSEK					
Limits for the number of standard resources (per system)	255	255	255	255	255	255
Limits for the number of internal resources (per system)	not limited by RTA-OSEK					
Limits for the number of nested resources (per system / per task)	255	255	255	255	255	255

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

4.2 Hardware Resources

4.2.1 ROM and RAM Overheads

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	58	58	58	58	58	58
	ROM	208	208	212	322	322	326
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	78	78	78	78	78	78
	ROM	284	284	288	398	398	402
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	108	108	108	108	108	108
	ROM	350	350	354	464	464	468
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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	44	44	44	44	44	44
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	48	48	48	48	48	48
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	56	64	n/a	56	64
ECC1, Integer task	RAM	n/a	n/a	n/a	68	68	68
	ROM	n/a	n/a	n/a	68	68	68
ECC1, floating-point task	RAM	n/a	n/a	n/a	92	92	92
	ROM	n/a	n/a	n/a	68	68	68
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	70
	ROM	n/a	n/a	n/a	n/a	n/a	76
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	94
	ROM	n/a	n/a	n/a	n/a	n/a	76
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	60	60	60	60	60	60
Category 2 ISR, floating-point	RAM	24	24	24	24	24	24
	ROM	80	80	80	80	80	80
Resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
Alarm	RAM	12	12	12	12	12	12
	ROM	40	40	40	40	40	40
Counter	RAM	4	4	4	4	4	4
	ROM	112	112	112	112	112	112
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	86	86	86	86	86	86
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	16	16	16	16	16	16
Arrivalpoint (writable)	RAM	16	16	16	16	16	16
	ROM	16	16	16	16	16	16
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	8	8	8	8	8	8
Taskset (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	64	64	64	64	64	64
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	72	80	n/a	72	80
ECC1, Integer task	RAM	n/a	n/a	n/a	80	80	80
	ROM	n/a	n/a	n/a	84	84	84
ECC1, floating-point task	RAM	n/a	n/a	n/a	104	104	104
	ROM	n/a	n/a	n/a	84	84	84

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	82
	ROM	n/a	n/a	n/a	n/a	n/a	92
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	106
	ROM	n/a	n/a	n/a	n/a	n/a	92
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	94	94	94	94	94	94
Category 2 ISR, floating-point	RAM	36	36	36	36	36	36
	ROM	110	110	110	110	110	110
Resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
Alarm	RAM	12	12	12	12	12	12
	ROM	40	40	40	40	40	40
Counter	RAM	4	4	4	4	4	4
	ROM	112	112	112	112	112	112
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	86	86	86	86	86	86
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	16	16	16	16	16	16
Arrivalpoint (writable)	RAM	16	16	16	16	16	16
	ROM	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	
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	8	8	8	8	8	8
Taskset (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	16	16	16	16	16	16
	ROM	72	72	72	72	72	72
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	72	72	72	72	72	72
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	80	88	n/a	80	88
ECC1, Integer task	RAM	n/a	n/a	n/a	84	84	84
	ROM	n/a	n/a	n/a	92	92	92
ECC1, floating-point task	RAM	n/a	n/a	n/a	108	108	108
	ROM	n/a	n/a	n/a	92	92	92
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	86
	ROM	n/a	n/a	n/a	n/a	n/a	100
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	110
	ROM	n/a	n/a	n/a	n/a	n/a	100
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	106	106	106	106	106	106
Category 2 ISR, floating-point	RAM	40	40	40	40	40	40
	ROM	122	122	122	122	122	122
Resource	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Linked resource	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Alarm	RAM	12	12	12	12	12	12
	ROM	44	44	44	44	44	44
Counter	RAM	4	4	4	4	4	4
	ROM	116	116	116	116	116	116
Message	RAM	11	11	11	31	31	31
	ROM	24	24	24	60	60	60
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
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	86	86	86	86	86	86
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	28	28	28	28	28	28
Arrivalpoint (writable)	RAM	28	28	28	28	28	28
	ROM	28	28	28	28	28	28
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	8	8	8	8	8	8
Taskset (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8

4.2.3 Size of Linkable Modules

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

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

The call variants are as follows:

Variant	Description
1i	Idle task is only ECC task.
CCCA	OSEK COM class.
CCCB	OSEK COM class.
CLEx	Resource tests in Extended OS Status.
fp	ECC task uses floating-point.
H	Used for heavyweight termination only.
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.

Variant	Description
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.
Task	Resource is used only by tasks.

Standard

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	166	262	316	186	290	398
	NS		140	234	288	160	258	370
	KL	2	118	214	268	138	238	346
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	22	22	22	22	22	22
ChainTask	SWL	1, 8	170	266	320	194	290	392
	SWH	1, 9	216	310	364	236	326	438
	NSL	8	170	266	320	194	290	392
	NSH	9	196	290	344	216	306	418
Schedule			138	138	196	138	138	196
GetTaskID			34	34	34	34	34	34
GetTaskState			94	94	94	122	122	122
EnableAllInterrupts			14	14	14	14	14	14
DisableAllInterrupts			22	22	22	22	22	22
ResumeAllInterrupts			30	30	30	30	30	30
SuspendAllInterrupts			48	48	48	48	48	48
ResumeOSInterrupts			30	30	30	30	30	30
SuspendOSInterrupts			42	42	42	42	42	42
GetResource	Task	7	86	86	112	86	86	112
	Combined	6	128	128	128	128	128	128
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	102	102	102	102	102	102
	Combined	6	194	194	194	194	194	194
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	176	176	330
	NS		n/a	n/a	n/a	148	148	298
	NS1i	10	n/a	n/a	n/a	50	n/a	n/a

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	KL	2	n/a	n/a	n/a	132	132	282
	KL1i	2, 10	n/a	n/a	n/a	26	n/a	n/a
ClearEvent			n/a	n/a	n/a	48	48	48
GetEvent			n/a	n/a	n/a	16	16	16
WaitEvent	<default>		n/a	n/a	n/a	230	230	454
	fp	11	n/a	n/a	n/a	258	258	510
	1i	10	n/a	n/a	n/a	24	n/a	n/a
GetAlarmBase			63	63	63	63	63	63
GetAlarm			90	90	90	90	90	90
SetRelAlarm			556	556	556	556	556	556
SetAbsAlarm			572	572	572	572	572	572
CancelAlarm			68	68	68	68	68	68
InitCounter			54	54	54	54	54	54
GetCounterValue			62	62	62	62	62	62
GetScheduleTableStatus		34	66	94	94	66	94	94
NextScheduleTable		34	80	232	232	80	232	232
StartScheduleTable		34	126	192	192	126	192	192
StopScheduleTable		34	92	136	136	92	136	136
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		10	10	10	10	10	10
ScheduleTable expiry point	Final		46	46	46	46	46	46
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	50	50	50	50	50	50
Process container	Non-Yielding	33	20	20	20	20	20	20
osek_tick_alarm	<default>		68	68	68	68	68	68
	KL	2	42	42	42	42	42	42
osek_incr_counter			40	40	40	40	40	40
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			170	170	170	170	170	170
ShutdownOS	NoHook	12	14	14	14	14	14	14
	Hook	13	28	28	28	28	28	28
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			26	26	26	26	26	26
StopCOM			24	24	24	24	24	24

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
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	64	64	64	224	224	224
	CCCB	15	224	224	224	224	224	224
GetMessageResource			42	42	42	42	42	42
ReleaseMessageResource			48	48	48	48	48	48
GetMessageStatus			56	56	56	56	56	56
SendMessage	SW CCCA	1, 14	92	92	92	256	256	256
	SW CCCB	1, 15	240	240	240	256	256	256
	NS CCCA	14	92	92	92	256	256	256
	NS CCCB	15	240	240	240	256	256	256
	KL CCCA	2, 14	72	72	72	232	232	232
	KL CCCB	2, 15	216	216	216	232	232	232
main_dispatch	NoHook	12	286	286	384	286	286	384
	Hook	13	320	320	392	320	320	392
sub_dispatch	B1LF	19	36	36	36	36	36	36
	B1HI	20	106	106	106	106	106	106
	B1HF	21	114	114	114	114	114	114
	B2LI	22	n/a	98	130	n/a	98	130
	B2LF	23	n/a	106	138	n/a	106	138
	B2HI	24	n/a	248	316	n/a	248	316
	B2HF	25	n/a	256	324	n/a	256	324
	E1HI	26	n/a	n/a	n/a	386	386	482
	E1HF	27	n/a	n/a	n/a	394	394	498
	E2HI	28	n/a	n/a	n/a	n/a	n/a	482
	E2HF	29	n/a	n/a	n/a	n/a	n/a	498
ErrorHook support		16	44	44	44	44	44	44
	ServiceID	17	52	52	52	52	52	52
	Parameters	18	72	72	72	72	72	72
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	160	352	426	208	434	556
	NS		132	324	386	190	402	526
	KL	2	112	304	364	156	362	486
ChainTaskset	SWL	1, 8	182	392	466	202	440	540
	SWH	1, 9	244	494	560	264	532	640

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	NSL	8	182	392	466	202	440	540
	NSH	9	224	474	540	244	512	620
GetTasksetRef			12	12	12	12	12	12
MergeTaskset			58	58	58	58	58	58
AssignTaskset			46	46	46	46	46	46
RemoveTaskset			64	64	64	64	64	64
TestSubTaskset			74	74	74	74	74	74
TestEquivalentTaskset			66	66	66	66	66	66
TickSchedule	SW	1	254	206	206	206	206	206
	NS		218	154	154	154	154	154
	KL	2	202	138	138	138	138	138
AdvanceSchedule	SW	1	242	190	190	190	190	190
	NS		206	138	138	138	138	138
	KL	2	190	122	122	122	122	122
StartSchedule			81	81	81	81	81	81
StopSchedule			56	56	56	56	56	56
GetScheduleStatus			96	96	96	96	96	96
GetScheduleValue			68	68	68	68	68	68
GetScheduleNext			16	16	16	16	16	16
SetScheduleNext			12	12	12	12	12	12
GetArrivalpointDelay			12	12	12	12	12	12
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			12	12	12	12	12	12
SetArrivalpointNext			8	8	8	8	8	8
TestArrivalpointWritable			32	32	32	32	32	32
GetExecutionTime			4	4	4	4	4	4
GetLargestExecutionTime			8	8	8	8	8	8
ResetLargestExecutionTime			4	4	4	4	4	4
GetStackOffset			26	26	26	26	26	26

Timing

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	166	262	316	186	290	398
	NS		140	234	288	160	258	370
	KL	2	118	214	268	138	238	346
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	22	22	22	22	22	22
ChainTask	SWL	1, 8	170	266	320	194	290	392
	SWH	1, 9	216	310	364	236	326	438
	NSL	8	170	266	320	194	290	392
	NSH	9	196	290	344	216	306	418
Schedule			160	160	218	160	160	218
GetTaskID			34	34	34	34	34	34
GetTaskState			94	94	94	122	122	122
EnableAllInterrupts			14	14	14	14	14	14
DisableAllInterrupts			22	22	22	22	22	22
ResumeAllInterrupts			30	30	30	30	30	30
SuspendAllInterrupts			48	48	48	48	48	48
ResumeOSInterrupts			30	30	30	30	30	30
SuspendOSInterrupts			42	42	42	42	42	42
GetResource	Task	7	86	86	112	86	86	112
	Combined	6	128	128	128	128	128	128
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	124	124	124	124	124	124
	Combined	6	232	232	232	232	232	232
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	176	176	330
	NS		n/a	n/a	n/a	148	148	298
	NS1i	10	n/a	n/a	n/a	50	n/a	n/a
	KL	2	n/a	n/a	n/a	132	132	282
	KL1i	2, 10	n/a	n/a	n/a	26	n/a	n/a
ClearEvent			n/a	n/a	n/a	48	48	48
GetEvent			n/a	n/a	n/a	16	16	16
WaitEvent	<default>		n/a	n/a	n/a	294	294	516
	fp	11	n/a	n/a	n/a	322	322	572

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	92	n/a	n/a
GetAlarmBase			63	63	63	63	63	63
GetAlarm			90	90	90	90	90	90
SetRelAlarm			556	556	556	556	556	556
SetAbsAlarm			572	572	572	572	572	572
CancelAlarm			68	68	68	68	68	68
InitCounter			54	54	54	54	54	54
GetCounterValue			62	62	62	62	62	62
GetScheduleTableStatus		34	66	94	94	66	94	94
NextScheduleTable		34	80	232	232	80	232	232
StartScheduleTable		34	126	192	192	126	192	192
StopScheduleTable		34	92	136	136	92	136	136
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		10	10	10	10	10	10
ScheduleTable expiry point	Final		46	46	46	46	46	46
GetISRID		4	36	36	36	36	36	36
Process container	Yielding	32	50	50	50	50	50	50
Process container	Non-Yielding	33	20	20	20	20	20	20
osek_tick_alarm	<default>		68	68	68	68	68	68
	KL	2	42	42	42	42	42	42
osek_incr_counter			40	40	40	40	40	40
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			222	222	222	222	222	222
ShutdownOS	NoHook	12	14	14	14	14	14	14
	Hook	13	28	28	28	28	28	28
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			26	26	26	26	26	26
StopCOM			24	24	24	24	24	24
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	64	64	64	224	224	224
	CCCB	15	224	224	224	224	224	224
GetMessageResource			42	42	42	42	42	42
ReleaseMessageResource			48	48	48	48	48	48

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			56	56	56	56	56	56
SendMessage	SW CCCA	1, 14	92	92	92	256	256	256
	SW CCCB	1, 15	240	240	240	256	256	256
	NS CCCA	14	92	92	92	256	256	256
	NS CCCB	15	240	240	240	256	256	256
	KL CCCA	2, 14	72	72	72	232	232	232
	KL CCCB	2, 15	216	216	216	232	232	232
main_dispatch	NoHook	12	354	354	428	354	354	428
	Hook	13	398	398	472	398	398	472
sub_dispatch	B1LF	19	20	20	20	20	20	20
	B1HI	20	112	112	112	112	112	112
	B1HF	21	120	120	120	120	120	120
	B2LI	22	n/a	78	110	n/a	78	110
	B2LF	23	n/a	86	118	n/a	86	118
	B2HI	24	n/a	252	320	n/a	252	320
	B2HF	25	n/a	260	328	n/a	260	328
	E1HI	26	n/a	n/a	n/a	432	432	524
	E1HF	27	n/a	n/a	n/a	440	440	540
	E2HI	28	n/a	n/a	n/a	n/a	n/a	524
	E2HF	29	n/a	n/a	n/a	n/a	n/a	540
ErrorHook support		16	44	44	44	44	44	44
	ServiceID	17	52	52	52	52	52	52
	Parameters	18	72	72	72	72	72	72
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	94	94	94	94	94	94
Timing_termination		4	74	74	74	74	74	74
ActivateTaskset	SW	1	160	352	426	208	434	556
	NS		132	324	386	190	402	526
	KL	2	112	304	364	156	362	486
ChainTaskset	SWL	1, 8	182	392	466	202	440	540
	SWH	1, 9	244	494	560	264	532	640
	NSL	8	182	392	466	202	440	540
	NSH	9	224	474	540	244	512	620
GetTasksetRef			12	12	12	12	12	12
MergeTaskset			58	58	58	58	58	58
AssignTaskset			46	46	46	46	46	46
RemoveTaskset			64	64	64	64	64	64

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			74	74	74	74	74	74
TestEquivalentTaskset			66	66	66	66	66	66
TickSchedule	SW	1	254	206	206	206	206	206
	NS		218	154	154	154	154	154
	KL	2	202	138	138	138	138	138
AdvanceSchedule	SW	1	242	190	190	190	190	190
	NS		206	138	138	138	138	138
	KL	2	190	122	122	122	122	122
StartSchedule			81	81	81	81	81	81
StopSchedule			56	56	56	56	56	56
GetScheduleStatus			96	96	96	96	96	96
GetScheduleValue			68	68	68	68	68	68
GetScheduleNext			16	16	16	16	16	16
SetScheduleNext			12	12	12	12	12	12
GetArrivalpointDelay			12	12	12	12	12	12
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			12	12	12	12	12	12
SetArrivalpointNext			8	8	8	8	8	8
TestArrivalpointWritable			32	32	32	32	32	32
GetExecutionTime			96	96	96	96	96	96
GetLargestExecutionTime			20	20	20	20	20	20
ResetLargestExecutionTime			16	16	16	16	16	16
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	220	326	374	244	356	476
	NS		314	424	468	340	452	562
	KL	2	182	286	332	210	316	434
TerminateTask	LExt	3	120	120	120	120	120	120

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	156	156	156	156	156	156
ChainTask	SWL	1, 8	270	386	434	296	408	512
	SWH	1, 9	326	426	472	352	456	564
	NSL	8	392	510	558	414	528	634
	NSH	9	438	524	572	454	556	676
Schedule			248	248	300	248	248	300
GetTaskID			54	54	54	54	54	54
GetTaskState			204	204	204	224	224	224
EnableAllInterrupts			34	34	34	34	34	34
DisableAllInterrupts			46	46	46	46	46	46
ResumeAllInterrupts			92	92	92	92	92	92
SuspendAllInterrupts			72	72	72	72	72	72
ResumeOSInterrupts			92	92	92	92	92	92
SuspendOSInterrupts			66	66	66	66	66	66
GetResource	Task	7	348	348	340	348	348	340
	Combined	6	338	338	338	338	338	338
	CLEx	3	312	312	312	312	312	312
ReleaseResource	Task	7	332	332	332	332	332	332
	Combined	6	430	430	430	430	430	430
	CLEx	3	268	268	268	268	268	268
SetEvent	SW	1	n/a	n/a	n/a	314	314	464
	NS		n/a	n/a	n/a	390	390	558
	NS1i	10	n/a	n/a	n/a	164	n/a	n/a
	KL	2	n/a	n/a	n/a	272	272	410
	KL1i	2, 10	n/a	n/a	n/a	156	n/a	n/a
ClearEvent			n/a	n/a	n/a	136	136	136
GetEvent			n/a	n/a	n/a	200	200	200
WaitEvent	<default>		n/a	n/a	n/a	414	414	620
	fp	11	n/a	n/a	n/a	442	442	676
	1i	10	n/a	n/a	n/a	214	n/a	n/a
GetAlarmBase			131	131	131	131	131	131
GetAlarm			132	132	132	132	132	132
SetRelAlarm			644	644	644	644	644	644
SetAbsAlarm			640	640	640	640	640	640
CancelAlarm			112	112	112	112	112	112
InitCounter			172	172	172	172	172	172
GetCounterValue			136	136	136	136	136	136

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	90	118	118	90	118	118
NextScheduleTable		34	104	256	256	104	256	256
StartScheduleTable		34	150	216	216	150	216	216
StopScheduleTable		34	116	160	160	116	160	160
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		10	10	10	10	10	10
ScheduleTable expiry point	Final		46	46	46	46	46	46
GetISRID		4	60	60	60	60	60	60
Process container	Yielding	32	50	50	50	50	50	50
Process container	Non-Yielding	33	20	20	20	20	20	20
osek_tick_alarm	<default>		68	68	68	68	68	68
	KL	2	42	42	42	42	42	42
osek_incr_counter			40	40	40	40	40	40
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			246	246	246	246	246	246
ShutdownOS	NoHook	12	26	26	26	26	26	26
	Hook	13	40	40	40	40	40	40
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			50	50	50	50	50	50
StopCOM			58	58	58	58	58	58
ReadFlag			38	38	38	38	38	38
ResetFlag			40	40	40	40	40	40
ReceiveMessage	CCCA	14	132	132	132	288	288	288
	CCCB	15	288	288	288	288	288	288
GetMessageResource			100	100	100	100	100	100
ReleaseMessageResource			108	108	108	108	108	108
GetMessageStatus			110	110	110	110	110	110
SendMessage	SW CCCA	1, 14	164	164	164	324	324	324
	SW CCCB	1, 15	308	308	308	324	324	324
	NS CCCA	14	164	164	164	324	324	324
	NS CCCB	15	308	308	308	324	324	324
	KL CCCA	2, 14	158	158	158	316	316	316
	KL CCCB	2, 15	298	298	298	316	316	316
main_dispatch	NoHook	12	354	354	428	354	354	428

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	398	398	472	398	398	472
sub_dispatch	B1LF	19	20	20	20	20	20	20
	B1HI	20	112	112	112	112	112	112
	B1HF	21	120	120	120	120	120	120
	B2LI	22	n/a	78	110	n/a	78	110
	B2LF	23	n/a	86	118	n/a	86	118
	B2HI	24	n/a	252	320	n/a	252	320
	B2HF	25	n/a	260	328	n/a	260	328
	E1HI	26	n/a	n/a	n/a	432	432	524
	E1HF	27	n/a	n/a	n/a	440	440	540
	E2HI	28	n/a	n/a	n/a	n/a	n/a	524
	E2HF	29	n/a	n/a	n/a	n/a	n/a	540
ErrorHook support		16	82	82	82	82	82	82
	ServiceID	17	92	92	92	92	92	92
	Parameters	18	118	118	118	118	118	118
validity_checks		3	23	23	23	23	23	23
Timing_dispatch		4	94	94	94	94	94	94
Timing_termination		4	78	78	78	78	78	78
ActivateTaskset	SW	1	306	426	488	372	532	628
	NS		386	524	584	476	632	740
	KL	2	274	378	440	328	470	576
ChainTaskset	SWL	1, 8	428	542	606	452	598	698
	SWH	1, 9	506	634	692	532	676	784
	NSL	8	554	676	738	578	728	824
	NSH	9	600	738	802	634	804	892
GetTasksetRef			130	130	130	130	130	130
MergeTaskset			238	238	238	238	238	238
AssignTaskset			220	220	220	220	220	220
RemoveTaskset			248	248	248	248	248	248
TestSubTaskset			256	256	256	256	256	256
TestEquivalentTaskset			252	252	252	252	252	252
TickSchedule	SW	1	390	296	296	296	296	296
	NS		474	402	402	402	402	402
	KL	2	334	242	242	242	242	242
AdvanceSchedule	SW	1	380	294	294	294	294	294
	NS		474	396	396	396	396	396
	KL	2	362	250	250	250	250	250

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			201	201	201	201	201	201
StopSchedule			142	142	142	142	142	142
GetScheduleStatus			182	182	182	182	182	182
GetScheduleValue			154	154	154	154	154	154
GetScheduleNext			102	102	102	102	102	102
SetScheduleNext			194	194	194	194	194	194
GetArrivalpointDelay			136	136	136	136	136	136
SetArrivalpointDelay			160	160	160	160	160	160
GetArrivalpointTasksetRef			132	132	132	132	132	132
GetArrivalpointNext			136	136	136	136	136	136
SetArrivalpointNext			210	210	210	210	210	210
TestArrivalpointWritable			154	154	154	154	154	154
GetExecutionTime			120	120	120	120	120	120
GetLargestExecutionTime			114	114	114	114	114	114
ResetLargestExecutionTime			108	108	108	108	108	108
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

4.3 Performance

The collection of performance data for the V850E/GreenHills port of the RTA-OSEK Component was achieved using a timer running eight times slower than the CPU clock speed. The figures in this section, therefore, have an uncertainty level of up to eight CPU cycles. The actual times are between 0 and eight 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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	127	199	255	119	199	287
	NS	103	175	231	103	183	271
	KL	103	175	223	95	175	271
TerminateTask	LExt	0	0	0	0	0	0
	H	143	135	151	143	143	143
ChainTask	SWL	327	415	543	391	463	623
	SWH	367	447	567	447	487	663
	NSL	319	415	535	399	463	623
	NSH	367	439	567	439	479	655
Schedule	SW	119	119	143	127	119	143
GetTaskID		39	47	39	47	39	47
GetTaskState		87	87	79	95	87	95
EnableAllInterrupts		23	15	23	23	15	23
DisableAllInterrupts		23	23	15	23	15	23
ResumeAllInterrupts		23	31	23	31	31	23
SuspendAllInterrupts		31	31	31	31	31	31
ResumeOSInterrupts		31	23	31	23	23	23
SuspendOSInterrupts		31	31	31	31	31	31
GetResource	Task	79	71	95	79	79	103
	Combined	63	71	63	63	63	71
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	63	63	63	63	63	63
	Combined	95	95	95	95	95	95
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	111	111	119
	NS	n/a	n/a	n/a	111	103	103
	KL	n/a	n/a	n/a	103	103	103
ClearEvent		n/a	n/a	n/a	47	55	47
GetEvent		n/a	n/a	n/a	39	39	39
WaitEvent	<default>	n/a	n/a	n/a	431	431	527
	fp	n/a	n/a	n/a	455	463	535
GetAlarmBase		71	71	79	71	71	63
GetAlarm		79	87	79	71	71	79

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		103	103	111	103	95	95
SetAbsAlarm		103	111	111	103	103	95
CancelAlarm		55	55	55	47	47	55
InitCounter		63	63	55	55	55	63
GetCounterValue		55	63	63	55	55	63
osek_tick_alarm	<default>	63	55	63	55	63	71
	KL	55	55	55	47	47	55
osek_incr_counter		23	23	15	15	31	23
GetActiveApplicationMode		15	15	7	15	15	15
StartOS		1919	2039	2039	1935	2047	1919
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	23	31	31	39	31	39
InitCOM		15	23	23	23	23	23
CloseCOM		15	15	15	23	23	23
StartCOM		39	55	47	151	143	135
StopCOM		31	23	23	39	31	39
ReadFlag		n/a	n/a	n/a	31	15	23
ResetFlag		n/a	n/a	n/a	23	23	23
ReceiveMessage		55	63	55	279	287	279
GetMessageResource		n/a	n/a	n/a	151	151	151
ReleaseMessageResource		n/a	n/a	n/a	127	127	127
GetMessageStatus		n/a	n/a	n/a	63	63	63
SendMessage	SW	191	271	319	439	503	583
	NS	175	255	295	423	495	575
	KL	167	239	287	391	463	575
ActivateTaskset	SW	71	1335	1391	87	1335	1631
	NS	55	1367	1359	79	1383	1479
	KL	39	1367	1367	71	1319	1543
	SW2	79	1335	1391	79	1335	1639
	NS2	55	1367	1367	79	1383	1487
	KL2	47	1359	1359	55	1327	1551
ChainTaskset	SWL	303	1503	1767	367	1623	1855
	SWH	351	1807	1871	423	1663	2031
	NSL	303	1559	1815	359	1631	1855
	NSH	335	1735	1807	407	1663	2015
GetTasksetRef		31	31	31	31	31	31

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		55	63	55	47	47	47
AssignTaskset		47	47	47	39	31	31
RemoveTaskset		55	55	63	47	47	47
TestSubTaskset		63	63	71	55	55	55
TestEquivalentTaskset		55	55	55	47	47	47
TickSchedule	SW	159	1503	1511	215	1479	1687
	NS	135	1487	1487	183	1455	1671
	KL	135	1471	1471	175	1447	1663
	SW2	151	1519	1511	207	1455	1679
	NS2	143	1479	1479	191	1439	1663
	KL2	127	1479	1479	183	1431	1655
AdvanceSchedule	SW	143	1479	1487	183	1463	1679
	NS	111	1471	1471	175	1439	1647
	KL	111	1455	1455	167	1423	1639
	SW2	143	1479	1479	191	1439	1663
	NS2	127	1463	1463	167	1423	1647
	KL2	111	1463	1455	151	1415	1639
StartSchedule		79	79	79	79	79	87
StopSchedule		71	71	79	79	79	71
GetScheduleStatus		79	71	71	71	71	71
GetScheduleValue		71	71	79	79	79	71
GetScheduleNext		31	39	39	31	31	31
SetScheduleNext		23	31	31	31	39	39
GetArrivalpointDelay		39	31	39	23	31	31
SetArrivalpointDelay		15	15	23	31	31	23
GetArrivalpointTasksetRef		15	15	15	31	23	31
GetArrivalpointNext		15	23	15	31	31	31
SetArrivalpointNext		15	23	23	31	31	23
TestArrivalpointWritable		39	39	39	31	31	31
GetExecutionTime		23	15	23	23	23	23
GetLargestExecutionTime		23	23	23	23	31	31
ResetLargestExecutionTime		23	15	15	23	23	23
GetStackOffset		31	23	23	31	23	23

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	127	199	255	127	191	287
	NS	103	191	223	111	183	271
	KL	95	175	215	111	175	263
TerminateTask	LExt	0	0	0	0	0	0
	H	383	351	383	367	383	375
ChainTask	SWL	591	695	815	671	751	903
	SWH	639	719	831	703	767	919
	NSL	599	679	807	671	751	911
	NSH	631	703	831	703	751	919
Schedule	SW	119	127	127	127	127	127
GetTaskID		47	39	39	39	47	39
GetTaskState		79	87	87	95	95	103
EnableAllInterrupts		23	23	23	15	15	23
DisableAllInterrupts		23	23	23	23	23	15
ResumeAllInterrupts		31	23	23	31	31	23
SuspendAllInterrupts		31	31	31	31	31	31
ResumeOSInterrupts		23	23	31	23	23	31
SuspendOSInterrupts		31	31	31	31	31	31
GetResource	Task	79	79	87	71	71	87
	Combined	63	71	63	71	63	71
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	63	63	63	63	63	63
	Combined	95	95	87	95	95	95
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	111	111	111
	NS	n/a	n/a	n/a	103	111	111
	KL	n/a	n/a	n/a	103	103	103
ClearEvent		n/a	n/a	n/a	47	47	55
GetEvent		n/a	n/a	n/a	31	39	31
WaitEvent	<default>	n/a	n/a	n/a	679	671	759
	fp	n/a	n/a	n/a	703	703	783
GetAlarmBase		71	63	71	71	63	71
GetAlarm		71	79	71	71	71	87

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		103	95	95	111	111	103
SetAbsAlarm		103	103	103	103	95	103
CancelAlarm		47	55	47	47	47	55
InitCounter		55	63	55	55	55	63
GetCounterValue		63	55	63	63	63	55
osek_tick_alarm	<default>	63	63	71	63	55	63
	KL	47	55	47	47	47	55
osek_incr_counter		23	15	23	31	23	23
GetActiveApplicationMode		15	15	23	15	15	15
StartOS		4823	5103	5095	5087	4815	5103
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	31	31	23	31	31	31
InitCOM		15	23	15	23	23	15
CloseCOM		15	23	15	23	23	15
StartCOM		55	55	47	143	143	151
StopCOM		39	31	31	23	31	39
ReadFlag		n/a	n/a	n/a	23	31	23
ResetFlag		n/a	n/a	n/a	15	15	23
ReceiveMessage		55	55	55	279	279	279
GetMessageResource		n/a	n/a	n/a	151	151	143
ReleaseMessageResource		n/a	n/a	n/a	119	127	127
GetMessageStatus		n/a	n/a	n/a	63	63	71
SendMessage	SW	191	263	319	439	503	591
	NS	167	263	295	423	495	559
	KL	159	247	279	399	463	559
ActivateTaskset	SW	71	1383	1327	79	1335	1631
	NS	63	1303	1311	71	1391	1487
	KL	47	1311	1295	63	1311	1543
	SW2	71	1391	1319	95	1343	1631
	NS2	55	1311	1311	63	1383	1495
	KL2	47	1295	1303	71	1327	1543
ChainTaskset	SWL	583	1839	2095	647	1903	2119
	SWH	615	1999	2079	695	1935	2303
	NSL	583	1759	2031	639	1911	2119
	NSH	615	2055	2135	679	1927	2287
GetTasksetRef		31	31	31	31	23	23

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		55	55	55	39	47	39
AssignTaskset		39	47	47	31	39	31
RemoveTaskset		55	55	55	47	47	47
TestSubTaskset		63	63	63	55	55	55
TestEquivalentTaskset		55	55	55	47	47	47
TickSchedule	SW	167	1439	1447	199	1471	1703
	NS	143	1423	1423	175	1455	1679
	KL	127	1423	1407	175	1447	1671
	SW2	159	1439	1455	207	1455	1687
	NS2	143	1415	1431	183	1439	1663
	KL2	127	1423	1415	175	1431	1663
AdvanceSchedule	SW	143	1423	1423	183	1455	1679
	NS	119	1415	1399	175	1439	1647
	KL	119	1391	1399	159	1439	1639
	SW2	143	1415	1423	183	1439	1671
	NS2	119	1407	1399	167	1423	1639
	KL2	111	1391	1407	159	1423	1631
StartSchedule		79	79	79	79	87	79
StopSchedule		71	79	71	79	71	79
GetScheduleStatus		71	71	71	71	71	71
GetScheduleValue		71	79	71	79	71	79
GetScheduleNext		39	39	39	31	31	31
SetScheduleNext		23	31	31	31	39	31
GetArrivalpointDelay		31	39	31	31	31	31
SetArrivalpointDelay		23	23	15	31	23	31
GetArrivalpointTasksetRef		23	15	15	31	31	31
GetArrivalpointNext		23	15	23	31	31	31
SetArrivalpointNext		23	23	23	23	23	23
TestArrivalpointWritable		39	39	39	31	31	31
GetExecutionTime		127	119	127	135	119	135
GetLargestExecutionTime		39	39	39	39	47	39
ResetLargestExecutionTime		39	39	39	39	39	39
GetStackOffset		31	31	31	31	23	23

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	423	511	551	423	503	583
	NS	479	583	607	487	575	639
	KL	399	495	543	407	495	559
TerminateTask	LExt	423	423	439	423	439	431
	H	447	455	463	463	455	471
ChainTask	SWL	927	1063	1183	1015	1135	1271
	SWH	991	1079	1175	1055	1143	1271
	NSL	1015	1143	1263	1111	1207	1367
	NSH	1071	1143	1255	1135	1207	1359
Schedule	SW	159	143	175	159	151	175
GetTaskID		55	47	39	55	47	47
GetTaskState		415	415	415	415	415	415
EnableAllInterrupts		23	23	23	23	23	23
DisableAllInterrupts		31	23	23	23	31	23
ResumeAllInterrupts		39	47	39	47	39	47
SuspendAllInterrupts		39	31	39	31	31	31
ResumeOSInterrupts		39	39	39	39	39	47
SuspendOSInterrupts		39	31	39	31	39	31
GetResource	Task	767	751	383	815	815	455
	Combined	295	295	287	359	359	351
	CLEx	367	383	367	439	447	431
ReleaseResource	Task	335	327	327	391	391	391
	Combined	287	287	287	351	359	351
	CLEx	327	327	335	391	391	391
SetEvent	SW	n/a	n/a	n/a	455	455	455
	NS	n/a	n/a	n/a	463	471	471
	KL	n/a	n/a	n/a	455	455	455
ClearEvent		n/a	n/a	n/a	95	87	79
GetEvent		n/a	n/a	n/a	423	423	423
WaitEvent	<default>	n/a	n/a	n/a	791	775	839
	fp	n/a	n/a	n/a	807	791	855
GetAlarmBase		239	239	255	239	239	263
GetAlarm		247	239	263	239	247	255

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		295	295	319	303	303	311
SetAbsAlarm		287	279	311	287	279	311
CancelAlarm		215	215	239	223	215	239
InitCounter		415	415	447	415	431	439
GetCounterValue		199	215	199	207	207	207
osek_tick_alarm	<default>	63	71	71	63	63	63
	KL	47	55	55	47	47	47
osek_incr_counter		23	23	23	31	23	15
GetActiveApplicationMode		15	23	7	15	15	15
StartOS		5391	5391	5095	5391	5391	5095
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	39	31	39	31	39	39
InitCOM		15	23	15	15	23	15
CloseCOM		23	31	15	15	23	15
StartCOM		63	47	55	151	159	151
StopCOM		39	39	47	39	39	47
ReadFlag		n/a	n/a	n/a	31	39	31
ResetFlag		n/a	n/a	n/a	31	39	31
ReceiveMessage		175	167	159	399	399	391
GetMessageResource		n/a	n/a	n/a	575	591	583
ReleaseMessageResource		n/a	n/a	n/a	535	527	535
GetMessageStatus		n/a	n/a	n/a	175	175	175
SendMessage	SW	607	695	727	839	903	999
	NS	671	767	791	911	983	1063
	KL	575	687	719	831	887	983
ActivateTaskset	SW	1239	2511	2583	1383	2647	2879
	NS	1359	2687	2631	1439	2639	2871
	KL	1223	2583	2503	1431	2591	2951
	SW2	1239	2519	2567	1383	2655	2879
	NS2	1359	2695	2631	1447	2639	2871
	KL2	1231	2567	2511	1439	2575	2951
ChainTaskset	SWL	1823	3127	3375	1927	3311	3367
	SWH	2063	3407	3487	2087	3343	3727
	NSL	1879	3319	3439	1975	3191	3575
	NSH	2055	3359	3743	2079	3671	3591
GetTasksetRef		359	367	359	359	367	367

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		127	127	119	135	135	135
AssignTaskset		111	111	111	119	127	119
RemoveTaskset		127	119	127	127	135	135
TestSubTaskset		127	127	127	135	135	135
TestEquivalentTaskset		119	127	127	135	127	135
TickSchedule	SW	247	2743	2695	1607	2783	3151
	NS	319	2823	2767	1687	2863	3231
	KL	223	2735	2663	1591	2775	3127
	SW2	247	2751	2687	1615	2759	3135
	NS2	319	2823	2759	1687	2831	3207
	KL2	223	2743	2663	1583	2751	3103
AdvanceSchedule	SW	239	2743	2671	1599	2767	3135
	NS	295	2807	2735	1663	2855	3207
	KL	231	2719	2663	1591	2759	3127
	SW2	239	2735	2671	1599	2743	3119
	NS2	295	2815	2735	1671	2823	3183
	KL2	231	2727	2663	1583	2735	3103
StartSchedule		143	143	143	143	143	143
StopSchedule		103	95	103	103	103	95
GetScheduleStatus		103	95	103	95	95	103
GetScheduleValue		95	103	95	95	95	95
GetScheduleNext		55	63	55	63	63	71
SetScheduleNext		127	127	127	119	119	119
GetArrivalpointDelay		79	71	79	79	79	87
SetArrivalpointDelay		103	103	103	95	95	95
GetArrivalpointTasksetRef		71	63	71	55	55	63
GetArrivalpointNext		63	71	71	55	55	63
SetArrivalpointNext		127	119	127	111	111	111
TestArrivalpointWritable		71	63	71	79	79	79
GetExecutionTime		135	127	119	135	127	119
GetLargestExecutionTime		343	343	343	343	343	343
ResetLargestExecutionTime		343	335	335	335	335	335
GetStackOffset		23	23	23	23	31	23

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	71	71	71	71	71	71
	Cat 2	79	111	111	111	111	111

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	71	71	71	71	71	71
	Cat 2	295	327	303	327	327	327

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	71	71	71	71	71	71
	Cat 2	295	327	327	327	303	303

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.

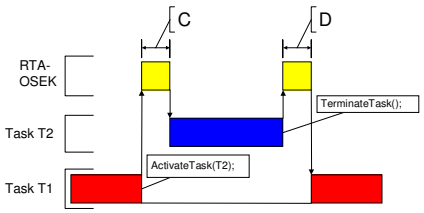

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

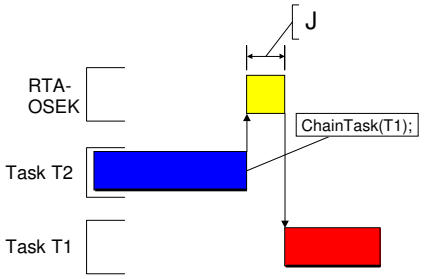

Figure 2: Task Chaining

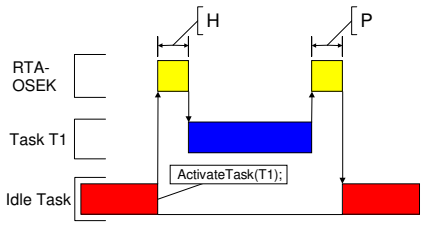

Figure 3: Task Activation from Idle Task

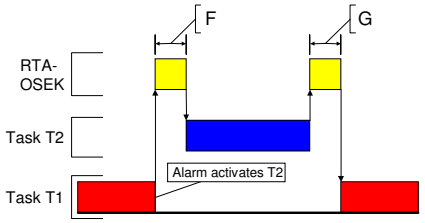

Figure 4: Task Activation from an Alarm

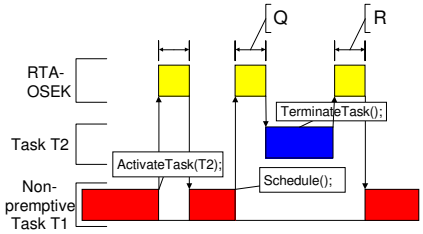

Figure 5: Non-Premptive Task Calls Schedule()

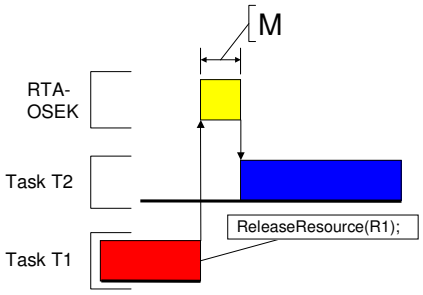

Figure 6: Blocked Task Activated by ReleaseResource()

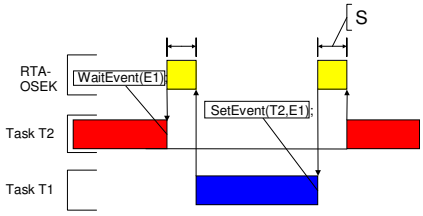

Figure 7: Waiting Task Activated by SetEvent()

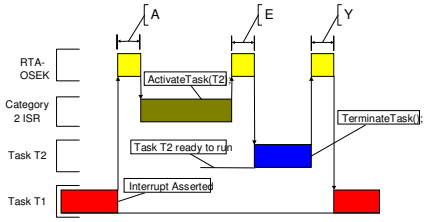

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	79	111	143	79	111	143
Figure 1: D	Heavy, Basic/Extended	135	159	191	183	191	223
ChainTask	Light, Basic	271	383	503	271	391	559
Figure 2: J	Heavy, Basic/Extended	543	655	791	575	671	871
Pre-emption	Light, Basic	231	335	463	231	335	511
Figure 1: C	Heavy, Basic/Extended	295	383	511	367	431	607
From idle task	Light, Basic	215	319	447	223	327	503
Figure 3: H	Heavy, Basic/Extended	279	383	503	359	423	607
Triggered by alarm	Light, Basic	295	399	535	295	407	575
Figure 4: F	Heavy, Basic/Extended	359	447	575	423	487	663
Schedule	Light, Basic	191	215	319	191	223	319
Figure 5: Q	Heavy, Basic/Extended	255	271	375	319	327	431
Release resource	Light, Basic	183	215	303	199	223	295
Figure 6: M	Heavy, Basic/Extended	255	263	351	327	327	415
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	559	575	775
From category 2 ISR	Light, Basic	175	223	303	207	223	311
Figure 8: E	Heavy, Basic/Extended	239	271	351	335	327	415

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	319	343	391	311	359	383
Figure 1: D	Heavy, Basic/Extended	375	391	431	415	415	455
ChainTask	Light, Basic	543	655	767	551	671	831
Figure 2: J	Heavy, Basic/Extended	1031	1135	1279	1087	1151	1359
Pre-emption	Light, Basic	431	535	663	431	527	687
Figure 1: C	Heavy, Basic/Extended	487	583	711	559	631	791
From idle task	Light, Basic	423	527	647	439	527	687
Figure 3: H	Heavy, Basic/Extended	487	583	703	567	615	807
Triggered by alarm	Light, Basic	503	615	719	503	591	759
Figure 4: F	Heavy, Basic/Extended	559	663	767	623	687	863
Schedule	Light, Basic	391	415	503	407	407	495
Figure 5: Q	Heavy, Basic/Extended	447	471	551	535	519	623
Release resource	Light, Basic	399	415	487	399	407	479
Figure 6: M	Heavy, Basic/Extended	455	463	535	535	519	599
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	743	727	919
From category 2 ISR	Light, Basic	631	655	735	655	655	727
Figure 8: E	Heavy, Basic/Extended	679	711	767	783	759	847

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	423	439	487	431	455	487
Figure 1: D	Heavy, Basic/Extended	455	479	511	503	527	543
ChainTask	Light, Basic	911	1039	1127	927	1031	1183
Figure 2: J	Heavy, Basic/Extended	1471	1575	1735	1535	1623	1775
Pre-emption	Light, Basic	727	815	927	727	831	983
Figure 1: C	Heavy, Basic/Extended	767	879	983	855	919	1087
From idle task	Light, Basic	711	823	943	727	831	983
Figure 3: H	Heavy, Basic/Extended	751	879	991	855	927	1079
Triggered by alarm	Light, Basic	807	903	1007	791	919	1063
Figure 4: F	Heavy, Basic/Extended	831	951	1063	911	999	1159
Schedule	Light, Basic	439	431	527	431	439	543
Figure 5: Q	Heavy, Basic/Extended	479	495	575	559	535	655
Release resource	Light, Basic	639	639	703	695	703	767
Figure 6: M	Heavy, Basic/Extended	687	703	759	823	815	879
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1103	1095	1287
From category 2 ISR	Light, Basic	623	655	751	655	679	759
Figure 8: E	Heavy, Basic/Extended	663	719	799	783	775	863

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	120	120	128	120	120	128
BCC1 lightweight, floating-point	128	128	136	128	128	136
BCC1 heavyweight, integer	188	188	196	188	188	196
BCC1 heavyweight, floating-point	188	188	196	188	188	196
BCC2 lightweight, integer	n/a	140	148	n/a	140	148
BCC2 lightweight, floating-point	n/a	140	148	n/a	140	148
BCC2 heavyweight, integer	n/a	196	204	n/a	196	204
BCC2 heavyweight, floating-point	n/a	196	204	n/a	196	204
ECC1 heavyweight, integer	n/a	n/a	n/a	216	216	224
ECC1 heavyweight, floating-point	n/a	n/a	n/a	216	216	224
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	224
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	224
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	112	112	112	112	112	112
BCC1 lightweight, floating-point	120	120	120	120	120	120
BCC1 heavyweight, integer	180	180	180	180	180	180
BCC1 heavyweight, floating-point	180	180	180	180	180	180
BCC2 lightweight, integer	n/a	132	132	n/a	132	132
BCC2 lightweight, floating-point	n/a	132	132	n/a	132	132
BCC2 heavyweight, integer	n/a	188	188	n/a	188	188
BCC2 heavyweight, floating-point	n/a	188	188	n/a	188	188
ECC1 heavyweight, integer	n/a	n/a	n/a	208	208	208
ECC1 heavyweight, floating-point	n/a	n/a	n/a	208	208	208
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	208
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	208

Timing

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Pre- and Post-Task hooks not used						
Task type						
BCC1 lightweight, integer	132	132	136	132	132	136
BCC1 lightweight, floating-point	136	136	140	136	136	140
BCC1 heavyweight, integer	200	200	204	200	200	204
BCC1 heavyweight, floating-point	200	200	204	200	200	204
BCC2 lightweight, integer	n/a	152	156	n/a	152	156
BCC2 lightweight, floating-point	n/a	152	156	n/a	152	156
BCC2 heavyweight, integer	n/a	208	212	n/a	208	212
BCC2 heavyweight, floating-point	n/a	208	212	n/a	208	212
ECC1 heavyweight, integer	n/a	n/a	n/a	228	228	232
ECC1 heavyweight, floating-point	n/a	n/a	n/a	228	228	232
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	232
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	232
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	132	132	136	132	132	136
BCC1 lightweight, floating-point	136	136	140	136	136	140
BCC1 heavyweight, integer	200	200	204	200	200	204
BCC1 heavyweight, floating-point	200	200	204	200	200	204
BCC2 lightweight, integer	n/a	152	156	n/a	152	156
BCC2 lightweight, floating-point	n/a	152	156	n/a	152	156
BCC2 heavyweight, integer	n/a	208	212	n/a	208	212
BCC2 heavyweight, floating-point	n/a	208	212	n/a	208	212
ECC1 heavyweight, integer	n/a	n/a	n/a	228	228	232
ECC1 heavyweight, floating-point	n/a	n/a	n/a	228	228	232
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	232
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	232

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	132	132	136	132	132	136
BCC1 lightweight, floating-point	136	136	140	136	136	140
BCC1 heavyweight, integer	200	200	204	200	200	204
BCC1 heavyweight, floating-point	200	200	204	200	200	204
BCC2 lightweight, integer	n/a	152	156	n/a	152	156
BCC2 lightweight, floating-point	n/a	152	156	n/a	152	156
BCC2 heavyweight, integer	n/a	208	212	n/a	208	212
BCC2 heavyweight, floating-point	n/a	208	212	n/a	208	212
ECC1 heavyweight, integer	n/a	n/a	n/a	236	236	240
ECC1 heavyweight, floating-point	n/a	n/a	n/a	236	236	240
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	240
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	240
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	132	132	136	132	132	136
BCC1 lightweight, floating-point	136	136	140	136	136	140
BCC1 heavyweight, integer	200	200	204	200	200	204
BCC1 heavyweight, floating-point	200	200	204	200	200	204
BCC2 lightweight, integer	n/a	152	156	n/a	152	156
BCC2 lightweight, floating-point	n/a	152	156	n/a	152	156
BCC2 heavyweight, integer	n/a	208	212	n/a	208	212
BCC2 heavyweight, floating-point	n/a	208	212	n/a	208	212
ECC1 heavyweight, integer	n/a	n/a	n/a	236	236	240
ECC1 heavyweight, floating-point	n/a	n/a	n/a	236	236	240
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	240
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	240

5 Compatibility with Pre-v5 Kernels

5.1 Updating the Application Version

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

5.2 32 Bit Timer Drivers

The v3.x kernel uses sixteen bit timer values, whereas the v5.0 kernel uses thirty-two bit timer values. Therefore any existing applications' timer drivers will need modifying. If the timer hardware supports only sixteen-bit timers, for example, the upper sixteen bits will need to be emulated in software. The example application demonstrates one method of achieving this for the Timer P hardware.

5.3 Vector Table

The vector table generated in the file `osvec1.850` is the same as the vector table that was previously generated in `osgen.850`. Simply assembling and linking `osvec1.850` in addition to `osgen.850` will allow existing applications' ISRs to continue working.

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.