RTA-OSEK

Binding Manual: V850EMultiLevel/GreenHills

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: RM-00085-02

Disclaimer

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

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

Trademarks

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

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

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

Contents

1 About this Guide

This guide provides target-specific information for the V850EMultiLevel/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 2)

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

Option	Description
`-O`	Use optimizations
`-noobj`	Turn off binary code generation
`-reserve_r2`	Do not use register r2 – reserved for the OS's exclusive use

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
`-registermode=32`	Use the full register set
`-reserve_r2`	Do not use register r2 – reserved for the OS's exclusive use
`-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 2)

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_jumptable	ROM	Jump table for indirect vector table
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 V850EMultiLevel/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	Hardware Priority	ISPR Values	Description
0	N/A	00000000	User level
1	7	10000000	Category 1 and 2 interrupts
2	6	x1000000	Category 1 and 2 interrupts
3	5	xx100000	Category 1 and 2 interrupts
4	4	xxx10000	Category 1 and 2 interrupts
5	3	xxxx1000	Category 1 and 2 interrupts
6	2	xxxxx100	Category 1 and 2 interrupts
7	1	xxxxxx10	Category 1 and 2 interrupts
8	0	xxxxxxx1	Category 1 and 2 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
`0x0080` to maximum vector for CPU variant	Category 1 or 2

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 CAG4-M, DJ3, FE2, FE3, FF2, FF3, FG2, FG3, FJ2, FJ3, FK3, PH2, PH3, PHO3, RS1, SG2, SG3, SJ2 and SJ3. These are in addition to the 'Generic V850E' variant. Further variants can be supported by contacting ETAS.

When RTA-OSEK generates an interrupt vector table for the V850, it only emits data for addresses 0x10 up to the *highest declared interrupt*. This allows RTA-OSEK to cope efficiently with chip variants with differently sized vector tables.

Reserved vectors

The RTA-OSEK component requires eight interrupt vectors to be reserved for its use. By default these will be generated in the vector table in the first eight unbound vectors starting from vector 0x80. This can be overridden by creating a dummy ISR in the RTA-OSEK GUI with the name '`reserved_os_vector`'. The first reserved vector will then be generated in its place and the remaining reserved vectors will be generated in the next seven unbound vectors.

The RTA-OSEK component also needs to know the addresses of the eight interrupt control registers (PICn) corresponding with the reserved vectors.

These should be mapped at link-time to the `_os_reserved_icr`*n* labels, where *n* ranges from 1 to 8. The linker directive file provided with the example application shows how this can be achieved.

3.1.6 Interrupt-handling assembler files

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:

1. If a Category 2 ISR is bound to the vector, the entry is a jump (`jr`) to the outer interrupt wrapper for that ISR.
2. If a Category 1 ISR is bound to the vector, the entry is a jump to the ISR's entry function.
3. If the vector is unbound, greater than or equal to vector 0x80 (or the vector for the dummy ISR `reserved_os_vector`, if it exists) and fewer than eight

entries have been reserved for the OS, then the entry is a jump to `os_reserved_vector` (see 'Reserved vectors' above for an explanation).

4. If the vector is unbound and there exists a default interrupt, the entry is a jump to the default interrupt's entry function.
5. 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, the mid-wrapper and the code for `os_reserved_vector`. 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:

6. If a Category 2 ISR is bound to the vector, the entry is the address of the outer interrupt wrapper for that ISR.
7. If a Category 1 ISR is bound to the vector, the entry is the address of the ISR's entry function.
8. If the vector is unbound and there exists a default interrupt, the entry is the address of the default interrupt's entry function.
9. 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.

Important: When using osvec3.850, the eight reserved vectors, each consisting of the following code, must be manually placed on the vector table. They may be placed anywhere, with the condition that the locations must match the `_os_reserved_icrn` addresses provided at link-time.

```
-- OS reserved vector
stsr  EIPC, r2
jmp   [r2]
.align     16
```

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

Vector Location	Label
0xVVVV	_os_wrapper_vvvv
e.g. 0x03A0	_os_wrapper_03a0

3.1.7 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.

3.1.8 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.

3.1.9 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 0x11223344556677899AA 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 0x11223344556677889AA at address 0x70.

Some V850 variants may contain other registers in flash after the security ID. For example, the CAG4-M variant contains the FMOP0 and FMOP1 registers at addresses 0x7A and 0x7B, respectively. These can be set simply by adding two (or more) additional bytes to the `OS_SECURITY_ID` macro. The provided example application demonstrates this by building osvec1.850 to write 0xDF to FMOP0, 0xFF to FMOP1, and to fill the remaining bytes from 0x7C to 0x7F with 0xFF.

3.1.10 Helper Macros

Category 1 ISR Macros

The macros `CAT1ISR_BEGIN()` and `CAT1ISR_END()` should be placed at the start and end of Category 1 ISR functions, to allow nested execution of interrupts by enabling/disabling interrupts and to perform some housekeeping required for the RTA-OSEK component to function correctly. A Category 1 ISR function should therefore look like the following code:

```
__interrupt void some_isr(void)
{
  CAT1ISR_BEGIN()
  /* user code goes here */
  CAT1ISR_END()
}
```

Enabling and Disabling Interrupt Sources

The macros `OS_ENABLE_INTERRUPT(PICn)` and `OS_DISABLE_INTERRUPT(PICn)` will enable/disable a specific interrupt source by writing to the provided interrupt control register. The arguments provided to these macros should be the register macros defined in the device header files provided by NEC. For example, the macro `PIC10` is defined in the file 'df3461.h' provided with the example application. The following code would disable, and then enable, the INTP10 interrupt.

```
/* disable INTP10 */
OS_DISABLE_INTERRUPT(PIC10);
/* enable INTP10 */
OS_ENABLE_INTERRUPT(PIC10);
```

3.2 Register Settings

The RTA-OSEK Component does not require the initialization of registers before calling `StartOS()`.

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

Register	Notes
r2	Reserved for the OS

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): 56

Timing

API max usage (bytes): 56

Extended

API max usage (bytes): 84

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

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/CAG4-M**
Clock speed (MHz)	80
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

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		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		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	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	76	76	76
	ROM	n/a	n/a	n/a	68	68	68
ECC1, floating-point task	RAM	n/a	n/a	n/a	78	78	78
	ROM	n/a	n/a	n/a	68	68	68
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	78
	ROM	n/a	n/a	n/a	n/a	n/a	76
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	80
	ROM	n/a	n/a	n/a	n/a	n/a	76
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	60	60	60	60	60	60
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	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	88	88	88
	ROM	n/a	n/a	n/a	84	84	84
ECC1, floating-point task	RAM	n/a	n/a	n/a	90	90	90
	ROM	n/a	n/a	n/a	84	84	84
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	90

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	92
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	92
	ROM	n/a	n/a	n/a	n/a	n/a	92
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	122	122	122	122	122	122
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	130	130	130	130	130	130
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
Schedule	RAM	16	16	16	16	16	16

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	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	92	92	92
	ROM	n/a	n/a	n/a	92	92	92
ECC1, floating-point task	RAM	n/a	n/a	n/a	94	94	94
	ROM	n/a	n/a	n/a	92	92	92
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	94
	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	96
	ROM	n/a	n/a	n/a	n/a	n/a	100
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	134	134	134	134	134	134
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	142	142	142	142	142	142
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
Linked resource	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
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.
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	238	336	382	258	358	460
	NS		210	308	358	234	334	436
	KL	2	114	214	264	138	236	338
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	234	322	376	254	344	444
	SWH	1, 9	270	360	414	290	380	484
	NSL	8	234	322	376	254	344	444
	NSH	9	252	342	396	272	362	466
Schedule			204	204	262	204	204	262
GetTaskID			32	32	32	32	32	32
GetTaskState			156	156	156	188	188	188
EnableAllInterrupts			84	84	84	84	84	84
DisableAllInterrupts			68	68	68	68	68	68
ResumeAllInterrupts			100	100	100	100	100	100
SuspendAllInterrupts			94	94	94	94	94	94
ResumeOSInterrupts			46	46	46	46	46	46
SuspendOSInterrupts			112	112	112	112	112	112
GetResource	Task	7	148	148	168	148	148	168
	Combined	6	300	300	300	300	300	300
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	164	164	164	164	164	164
	Combined	6	326	326	326	326	326	326
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	238	238	374
	NS		n/a	n/a	n/a	210	210	346
	NS1i	10	n/a	n/a	n/a	112	n/a	n/a
	KL	2	n/a	n/a	n/a	128	128	272
	KL1i	2, 10	n/a	n/a	n/a	26	n/a	n/a
ClearEvent			n/a	n/a	n/a	112	112	112

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	16	16	16
WaitEvent	<default>		n/a	n/a	n/a	346	346	682
	fp	11	n/a	n/a	n/a	374	374	740
	1i	10	n/a	n/a	n/a	24	n/a	n/a
GetAlarmBase			119	119	119	119	119	119
GetAlarm			146	146	146	146	146	146
SetRelAlarm			628	628	628	628	628	628
SetAbsAlarm			650	650	650	650	650	650
CancelAlarm			128	128	128	128	128	128
InitCounter			114	114	114	114	114	114
GetCounterValue			122	122	122	122	122	122
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>		128	128	128	128	128	128
	KL	2	42	42	42	42	42	42
osek_incr_counter			42	42	42	42	42	42
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			286	286	286	286	286	286
ShutdownOS	NoHook	12	84	84	84	84	84	84
	Hook	13	94	94	94	94	94	94
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	122	122	122	280	280	280

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	280	280	280	280	280	280
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	152	152	152	310	310	310
	SW CCCB	1, 15	294	294	294	310	310	310
	NS CCCA	14	152	152	152	310	310	310
	NS CCCB	15	294	294	294	310	310	310
	KL CCCA	2, 14	72	72	72	230	230	230
	KL CCCB	2, 15	214	214	214	230	230	230
main_dispatch	NoHook	12	340	340	410	340	340	410
	Hook	13	384	384	452	384	384	452
sub_dispatch	B1LF	19	36	36	36	36	36	36
	B1HI	20	162	162	162	162	162	162
	B1HF	21	170	170	170	170	170	170
	B2LI	22	n/a	154	186	n/a	154	186
	B2LF	23	n/a	162	194	n/a	162	194
	B2HI	24	n/a	464	532	n/a	464	532
	B2HF	25	n/a	472	540	n/a	472	540
	E1HI	26	n/a	n/a	n/a	620	620	712
	E1HF	27	n/a	n/a	n/a	628	628	720
	E2HI	28	n/a	n/a	n/a	n/a	n/a	712
	E2HF	29	n/a	n/a	n/a	n/a	n/a	720
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	228	408	472	276	480	576
	NS		204	384	448	248	452	552
	KL	2	108	288	348	152	360	464
ChainTaskset	SWL	1, 8	236	422	484	258	470	564
	SWH	1, 9	296	512	572	318	564	656
	NSL	8	236	422	484	258	470	564
	NSH	9	278	494	554	300	546	638
GetTasksetRef			12	12	12	12	12	12

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
MergeTaskset			118	118	118	118	118	118
AssignTaskset			106	106	106	106	106	106
RemoveTaskset			122	122	122	122	122	122
TestSubTaskset			132	132	132	132	132	132
TestEquivalentTaskset			128	128	128	128	128	128
TickSchedule	SW	1	318	274	274	274	274	274
	NS		282	222	222	222	222	222
	KL	2	202	138	138	138	138	138
AdvanceSchedule	SW	1	302	250	250	250	250	250
	NS		270	202	202	202	202	202
	KL	2	190	122	122	122	122	122
StartSchedule			143	143	143	143	143	143
StopSchedule			118	118	118	118	118	118
GetScheduleStatus			158	158	158	158	158	158
GetScheduleValue			130	130	130	130	130	130
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			28	28	28	28	28	28

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	238	336	382	258	358	460
	NS		210	308	358	234	334	436
	KL	2	114	214	264	138	236	338
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	234	322	376	254	344	444
	SWH	1, 9	270	360	414	290	380	484
	NSL	8	234	322	376	254	344	444
	NSH	9	252	342	396	272	362	466
Schedule			226	226	284	226	226	284
GetTaskID			32	32	32	32	32	32
GetTaskState			156	156	156	188	188	188
EnableAllInterrupts			84	84	84	84	84	84
DisableAllInterrupts			68	68	68	68	68	68
ResumeAllInterrupts			100	100	100	100	100	100
SuspendAllInterrupts			94	94	94	94	94	94
ResumeOSInterrupts			46	46	46	46	46	46
SuspendOSInterrupts			112	112	112	112	112	112
GetResource	Task	7	148	148	168	148	148	168
	Combined	6	300	300	300	300	300	300
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	186	186	186	186	186	186
	Combined	6	370	370	370	370	370	370
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	238	238	374
	NS		n/a	n/a	n/a	210	210	346
	NS1i	10	n/a	n/a	n/a	112	n/a	n/a
	KL	2	n/a	n/a	n/a	128	128	272
	KL1i	2, 10	n/a	n/a	n/a	26	n/a	n/a
ClearEvent			n/a	n/a	n/a	112	112	112
GetEvent			n/a	n/a	n/a	16	16	16
WaitEvent	<default>		n/a	n/a	n/a	462	462	794
	fp	11	n/a	n/a	n/a	490	490	852

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	154	n/a	n/a
GetAlarmBase			119	119	119	119	119	119
GetAlarm			146	146	146	146	146	146
SetRelAlarm			628	628	628	628	628	628
SetAbsAlarm			650	650	650	650	650	650
CancelAlarm			128	128	128	128	128	128
InitCounter			114	114	114	114	114	114
GetCounterValue			122	122	122	122	122	122
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>		128	128	128	128	128	128
	KL	2	42	42	42	42	42	42
osek_incr_counter			42	42	42	42	42	42
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			342	342	342	342	342	342
ShutdownOS	NoHook	12	84	84	84	84	84	84
	Hook	13	94	94	94	94	94	94
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	122	122	122	280	280	280
	CCCB	15	280	280	280	280	280	280
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	152	152	152	310	310	310
	SW CCCB	1, 15	294	294	294	310	310	310
	NS CCCA	14	152	152	152	310	310	310
	NS CCCB	15	294	294	294	310	310	310
	KL CCCA	2, 14	72	72	72	230	230	230
	KL CCCB	2, 15	214	214	214	230	230	230
main_dispatch	NoHook	12	346	346	420	346	346	420
	Hook	13	388	388	464	388	388	464
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	310	378	n/a	310	378
	B2HF	25	n/a	318	386	n/a	318	386
	E1HI	26	n/a	n/a	n/a	536	536	628
	E1HF	27	n/a	n/a	n/a	544	544	636
	E2HI	28	n/a	n/a	n/a	n/a	n/a	628
	E2HF	29	n/a	n/a	n/a	n/a	n/a	636
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	98	98	98	98	98	98
Timing_termination		4	136	136	136	136	136	136
ActivateTaskset	SW	1	228	408	472	276	480	576
	NS		204	384	448	248	452	552
	KL	2	108	288	348	152	360	464
ChainTaskset	SWL	1, 8	236	422	484	258	470	564
	SWH	1, 9	296	512	572	318	564	656
	NSL	8	236	422	484	258	470	564
	NSH	9	278	494	554	300	546	638
GetTasksetRef			12	12	12	12	12	12
MergeTaskset			118	118	118	118	118	118
AssignTaskset			106	106	106	106	106	106
RemoveTaskset			122	122	122	122	122	122

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			132	132	132	132	132	132
TestEquivalentTaskset			128	128	128	128	128	128
TickSchedule	SW	1	318	274	274	274	274	274
	NS		282	222	222	222	222	222
	KL	2	202	138	138	138	138	138
AdvanceSchedule	SW	1	302	250	250	250	250	250
	NS		270	202	202	202	202	202
	KL	2	190	122	122	122	122	122
StartSchedule			143	143	143	143	143	143
StopSchedule			118	118	118	118	118	118
GetScheduleStatus			158	158	158	158	158	158
GetScheduleValue			130	130	130	130	130	130
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			160	160	160	160	160	160
GetLargestExecutionTime			20	20	20	20	20	20
ResetLargestExecutionTime			16	16	16	16	16	16
GetStackOffset			28	28	28	28	28	28

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	364	462	512	382	482	580
	NS		458	560	608	484	580	666
	KL	2	208	304	354	232	324	416
TerminateTask	LExt	3	132	132	132	132	132	132

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	158	158	158	158	158	158
ChainTask	SWL	1, 8	392	504	550	410	524	620
	SWH	1, 9	454	548	594	478	566	672
	NSL	8	508	632	678	528	648	746
	NSH	9	554	660	710	580	674	778
Schedule			386	386	444	386	386	444
GetTaskID			52	52	52	52	52	52
GetTaskState			330	330	330	348	348	348
EnableAllInterrupts			104	104	104	104	104	104
DisableAllInterrupts			106	106	106	106	106	106
ResumeAllInterrupts			156	156	156	156	156	156
SuspendAllInterrupts			132	132	132	132	132	132
ResumeOSInterrupts			108	108	108	108	108	108
SuspendOSInterrupts			158	158	158	158	158	158
GetResource	Task	7	480	480	474	480	480	474
	Combined	6	562	562	562	562	562	562
	CLEx	3	430	430	430	430	430	430
ReleaseResource	Task	7	488	488	488	488	488	488
	Combined	6	680	680	680	680	680	680
	CLEx	3	408	408	408	408	408	408
SetEvent	SW	1	n/a	n/a	n/a	444	444	598
	NS		n/a	n/a	n/a	530	530	670
	NS1i	10	n/a	n/a	n/a	306	n/a	n/a
	KL	2	n/a	n/a	n/a	282	282	418
	KL1i	2, 10	n/a	n/a	n/a	174	n/a	n/a
ClearEvent			n/a	n/a	n/a	220	220	220
GetEvent			n/a	n/a	n/a	190	190	190
WaitEvent	<default>		n/a	n/a	n/a	612	612	944
	fp	11	n/a	n/a	n/a	642	642	1006
	1i	10	n/a	n/a	n/a	294	n/a	n/a
GetAlarmBase			263	263	263	263	263	263
GetAlarm			258	258	258	258	258	258
SetRelAlarm			782	782	782	782	782	782
SetAbsAlarm			786	786	786	786	786	786
CancelAlarm			238	238	238	238	238	238
InitCounter			298	298	298	298	298	298
GetCounterValue			272	272	272	272	272	272

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>		194	194	194	194	194	194
	KL	2	42	42	42	42	42	42
osek_incr_counter			42	42	42	42	42	42
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			378	378	378	378	378	378
ShutdownOS	NoHook	12	116	116	116	116	116	116
	Hook	13	126	126	126	126	126	126
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	258	258	258	414	414	414
	CCCB	15	414	414	414	414	414	414
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	302	302	302	458	458	458
	SW CCCB	1, 15	442	442	442	458	458	458
	NS CCCA	14	302	302	302	458	458	458
	NS CCCB	15	442	442	442	458	458	458
	KL CCCA	2, 14	178	178	178	336	336	336
	KL CCCB	2, 15	318	318	318	336	336	336
main_dispatch	NoHook	12	346	346	420	346	346	420

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	388	388	464	388	388	464
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	326	394	n/a	326	394
	B2HF	25	n/a	334	402	n/a	334	402
	E1HI	26	n/a	n/a	n/a	574	574	666
	E1HF	27	n/a	n/a	n/a	582	582	674
	E2HI	28	n/a	n/a	n/a	n/a	n/a	666
	E2HF	29	n/a	n/a	n/a	n/a	n/a	674
ErrorHook support		16	176	176	176	176	176	176
	ServiceID	17	186	186	186	186	186	186
	Parameters	18	212	212	212	212	212	212
validity_checks		3	25	25	25	25	25	25
Timing_dispatch		4	98	98	98	98	98	98
Timing_termination		4	158	158	158	158	158	158
ActivateTaskset	SW	1	450	550	612	482	616	728
	NS		542	642	700	574	706	820
	KL	2	282	370	436	322	442	516
ChainTaskset	SWL	1, 8	522	644	704	544	672	780
	SWH	1, 9	608	736	796	642	776	894
	NSL	8	666	756	818	686	824	914
	NSH	9	732	834	894	746	902	998
GetTasksetRef			130	130	130	130	130	130
MergeTaskset			378	378	378	378	378	378
AssignTaskset			348	348	348	348	348	348
RemoveTaskset			382	382	382	382	382	382
TestSubTaskset			394	394	394	394	394	394
TestEquivalentTaskset			390	390	390	390	390	390
TickSchedule	SW	1	502	420	420	420	420	420
	NS		602	534	534	534	534	534
	KL	2	358	264	264	264	264	264
AdvanceSchedule	SW	1	500	418	418	418	418	418
	NS		590	522	522	522	522	522
	KL	2	364	266	266	266	266	266

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			341	341	341	341	341	341
StopSchedule			282	282	282	282	282	282
GetScheduleStatus			322	322	322	322	322	322
GetScheduleValue			282	282	282	282	282	282
GetScheduleNext			98	98	98	98	98	98
SetScheduleNext			190	190	190	190	190	190
GetArrivalpointDelay			138	138	138	138	138	138
SetArrivalpointDelay			156	156	156	156	156	156
GetArrivalpointTasksetRef			134	134	134	134	134	134
GetArrivalpointNext			138	138	138	138	138	138
SetArrivalpointNext			202	202	202	202	202	202
TestArrivalpointWritable			150	150	150	150	150	150
GetExecutionTime			246	246	246	246	246	246
GetLargestExecutionTime			114	114	114	114	114	114
ResetLargestExecutionTime			108	108	108	108	108	108
GetStackOffset			28	28	28	28	28	28

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 V850EMultiLevel/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		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	195	287	339	207	275	379
	NS	187	279	327	203	271	367
	KL	87	175	223	99	163	263
TerminateTask	LExt	0	0	0	0	0	0
	H	235	235	247	243	243	247
ChainTask	SWL	487	591	719	567	651	811
	SWH	643	727	843	723	779	939
	NSL	483	599	727	559	647	807
	NSH	631	723	839	711	771	931
Schedule	SW	203	199	227	203	203	231
GetTaskID		39	35	43	43	43	39
GetTaskState		163	167	171	183	187	179
EnableAllInterrupts		71	71	75	75	79	79
DisableAllInterrupts		91	91	95	87	95	95
ResumeAllInterrupts		79	75	79	79	83	79
SuspendAllInterrupts		99	99	103	103	107	107
ResumeOSInterrupts		71	75	79	79	83	79
SuspendOSInterrupts		95	99	103	103	103	107
GetResource	Task	163	171	183	175	171	187
	Combined	179	175	187	179	187	183
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	147	151	155	155	151	151
	Combined	339	335	331	339	339	343
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	207	199	207
	NS	n/a	n/a	n/a	195	203	199
	KL	n/a	n/a	n/a	99	103	111
ClearEvent		n/a	n/a	n/a	187	187	195
GetEvent		n/a	n/a	n/a	39	39	39
WaitEvent	<default>	n/a	n/a	n/a	619	603	731
	fp	n/a	n/a	n/a	631	623	775
GetAlarmBase		155	151	159	159	159	151
GetAlarm		155	155	159	167	171	167

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		195	191	199	199	195	199
SetAbsAlarm		183	191	195	195	195	187
CancelAlarm		139	143	143	143	147	147
InitCounter		143	147	139	151	143	151
GetCounterValue		147	143	151	147	147	143
osek_tick_alarm	<default>	151	147	151	155	155	155
	KL	51	47	51	47	47	51
osek_incr_counter		11	19	23	23	23	15
GetActiveApplicationMode		11	11	15	15	15	7
StartOS		2103	2211	2211	2211	2211	2103
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	131	127	131	131	127	127
InitCOM		15	19	23	23	15	15
CloseCOM		19	19	19	15	19	15
StartCOM		39	35	47	147	147	139
StopCOM		27	27	31	31	35	31
ReadFlag		n/a	n/a	n/a	23	19	23
ResetFlag		n/a	n/a	n/a	15	23	23
ReceiveMessage		139	143	139	379	371	367
GetMessageResource		n/a	n/a	n/a	243	235	231
ReleaseMessageResource		n/a	n/a	n/a	215	211	211
GetMessageStatus		n/a	n/a	n/a	67	67	59
SendMessage	SW	359	447	503	599	675	779
	NS	355	431	491	599	663	751
	KL	147	239	283	391	463	563
ActivateTaskset	SW	155	1415	1611	171	1551	1655
	NS	147	1467	1535	163	1551	1711
	KL	35	1371	1367	55	1443	1663
	SW2	155	1415	1607	167	1559	1647
	NS2	147	1459	1527	155	1551	1707
	KL2	43	1371	1375	55	1443	1671
ChainTaskset	SWL	467	1731	1819	543	1735	1911
	SWH	615	1883	2027	687	1951	2047
	NSL	467	1731	1879	539	1735	1975
	NSH	603	1943	2019	683	2011	2047
GetTasksetRef		31	27	31	31	35	31

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		139	135	131	139	143	139
AssignTaskset		123	127	127	127	119	123
RemoveTaskset		135	135	135	139	139	131
TestSubTaskset		139	139	139	139	135	135
TestEquivalentTaskset		127	127	139	139	139	139
TickSchedule	SW	231	1607	1607	283	1695	1907
	NS	219	1587	1591	275	1679	1895
	KL	123	1483	1487	167	1567	1783
	SW2	239	1603	1599	283	1679	1895
	NS2	223	1587	1591	275	1663	1891
	KL2	127	1483	1487	171	1559	1783
AdvanceSchedule	SW	219	1579	1579	263	1659	1883
	NS	203	1567	1567	251	1651	1871
	KL	99	1463	1467	151	1547	1767
	SW2	215	1579	1579	263	1643	1875
	NS2	199	1567	1567	251	1643	1867
	KL2	107	1463	1467	143	1531	1759
StartSchedule		171	167	171	175	175	175
StopSchedule		155	155	159	159	159	167
GetScheduleStatus		167	155	159	159	167	163
GetScheduleValue		151	155	159	159	163	167
GetScheduleNext		27	27	31	31	23	27
SetScheduleNext		27	27	31	31	31	31
GetArrivalpointDelay		27	23	27	27	27	35
SetArrivalpointDelay		23	19	23	23	23	27
GetArrivalpointTasksetRef		15	19	23	23	23	19
GetArrivalpointNext		23	11	15	15	15	27
SetArrivalpointNext		23	19	23	23	23	27
TestArrivalpointWritable		23	31	35	35	35	27
GetExecutionTime		19	15	15	19	19	19
GetLargestExecutionTime		23	23	27	31	23	27
ResetLargestExecutionTime		23	23	27	23	23	27
GetStackOffset		27	31	31	35	35	31

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	203	291	335	211	275	379
	NS	191	275	327	191	267	363
	KL	91	179	223	87	151	255
TerminateTask	LExt	0	0	0	0	0	0
	H	567	563	571	559	563	563
ChainTask	SWL	831	915	1051	883	983	1143
	SWH	1011	1091	1211	1083	1143	1283
	NSL	831	931	1039	903	979	1151
	NSH	999	1083	1203	1075	1139	1295
Schedule	SW	207	207	227	199	203	227
GetTaskID		43	43	47	39	35	39
GetTaskState		167	167	167	183	183	187
EnableAllInterrupts		75	75	75	67	67	79
DisableAllInterrupts		87	95	95	91	91	99
ResumeAllInterrupts		83	83	79	79	75	83
SuspendAllInterrupts		103	103	103	99	103	103
ResumeOSInterrupts		83	83	79	79	75	83
SuspendOSInterrupts		103	103	103	103	103	107
GetResource	Task	167	167	183	175	171	191
	Combined	183	183	179	183	175	187
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	155	155	159	155	147	159
	Combined	343	343	339	331	339	335
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	199	195	199
	NS	n/a	n/a	n/a	199	199	207
	KL	n/a	n/a	n/a	99	99	103
ClearEvent		n/a	n/a	n/a	187	187	191
GetEvent		n/a	n/a	n/a	35	35	39
WaitEvent	<default>	n/a	n/a	n/a	891	895	995
	fp	n/a	n/a	n/a	923	923	1003
GetAlarmBase		159	151	155	155	155	151
GetAlarm		167	167	159	159	163	167

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		191	191	195	187	195	199
SetAbsAlarm		195	195	187	191	191	187
CancelAlarm		147	147	147	143	143	139
InitCounter		151	151	143	143	143	151
GetCounterValue		143	143	147	147	147	143
osek_tick_alarm	<default>	155	155	151	151	151	155
	KL	47	47	51	47	51	55
osek_incr_counter		23	23	15	19	19	23
GetActiveApplicationMode		7	7	15	3	3	15
StartOS		5003	5003	5279	5279	5003	5279
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	127	135	135	131	131	127
InitCOM		23	23	15	19	19	15
CloseCOM		23	23	15	19	19	15
StartCOM		47	47	43	143	151	147
StopCOM		31	31	31	27	27	31
ReadFlag		n/a	n/a	n/a	19	19	19
ResetFlag		n/a	n/a	n/a	15	15	19
ReceiveMessage		139	147	143	363	375	367
GetMessageResource		n/a	n/a	n/a	239	235	243
ReleaseMessageResource		n/a	n/a	n/a	211	215	215
GetMessageStatus		n/a	n/a	n/a	63	55	67
SendMessage	SW	367	455	491	607	671	775
	NS	359	439	487	591	655	759
	KL	159	243	279	391	459	563
ActivateTaskset	SW	159	1423	1539	167	1555	1655
	NS	155	1475	1599	159	1547	1707
	KL	43	1367	1435	55	1431	1667
	SW2	159	1419	1539	167	1555	1655
	NS2	151	1471	1599	155	1547	1711
	KL2	51	1375	1427	51	1439	1663
ChainTaskset	SWL	815	2075	2199	867	2063	2239
	SWH	995	2247	2451	1063	2315	2415
	NSL	815	2075	2143	867	2075	2299
	NSH	983	2287	2447	1051	2371	2415
GetTasksetRef		23	31	35	27	27	31

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		139	139	143	135	135	139
AssignTaskset		123	123	127	119	123	123
RemoveTaskset		131	131	139	127	127	131
TestSubTaskset		135	135	143	131	139	135
TestEquivalentTaskset		139	139	131	135	135	139
TickSchedule	SW	243	1611	1663	287	1691	1907
	NS	231	1595	1651	275	1675	1899
	KL	127	1487	1547	159	1567	1783
	SW2	243	1611	1663	287	1675	1899
	NS2	231	1595	1651	275	1659	1887
	KL2	131	1491	1539	167	1555	1775
AdvanceSchedule	SW	223	1583	1639	259	1663	1883
	NS	211	1571	1631	251	1651	1867
	KL	107	1467	1527	147	1551	1763
	SW2	223	1579	1639	259	1647	1875
	NS2	211	1571	1623	251	1639	1859
	KL2	107	1467	1527	147	1531	1763
StartSchedule		175	167	175	163	171	175
StopSchedule		167	167	159	163	155	167
GetScheduleStatus		167	167	171	167	163	163
GetScheduleValue		163	163	155	155	159	167
GetScheduleNext		31	23	31	23	19	27
SetScheduleNext		23	31	31	27	27	31
GetArrivalpointDelay		31	27	31	31	23	35
SetArrivalpointDelay		27	23	27	23	19	27
GetArrivalpointTasksetRef		23	23	19	15	19	19
GetArrivalpointNext		23	15	27	23	11	27
SetArrivalpointNext		15	23	27	23	19	27
TestArrivalpointWritable		35	35	27	23	31	27
GetExecutionTime		219	223	223	219	211	211
GetLargestExecutionTime		47	43	47	39	43	47
ResetLargestExecutionTime		39	43	43	39	39	39
GetStackOffset		35	35	31	27	27	35

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	587	687	719	599	671	747
	NS	659	763	791	663	747	823
	KL	431	527	555	439	519	587
TerminateTask	LExt	743	755	755	747	759	763
	H	767	779	787	787	795	791
ChainTask	SWL	1351	1471	1579	1423	1527	1675
	SWH	1535	1639	1747	1631	1699	1815
	NSL	1423	1547	1639	1503	1607	1755
	NSH	1619	1707	1807	1707	1763	1903
Schedule	SW	327	331	339	315	323	351
GetTaskID		51	47	43	43	51	51
GetTaskState		599	599	595	595	595	599
EnableAllInterrupts		79	79	79	75	79	79
DisableAllInterrupts		99	99	103	99	107	107
ResumeAllInterrupts		95	95	91	91	95	95
SuspendAllInterrupts		115	115	111	115	115	115
ResumeOSInterrupts		95	95	91	91	95	95
SuspendOSInterrupts		115	115	115	111	115	115
GetResource	Task	891	891	527	951	959	591
	Combined	423	423	423	491	487	487
	CLEx	507	507	511	575	579	579
ReleaseResource	Task	507	507	507	571	575	579
	Combined	619	619	619	679	683	683
	CLEx	487	487	483	551	547	555
SetEvent	SW	n/a	n/a	n/a	619	623	639
	NS	n/a	n/a	n/a	651	651	655
	KL	n/a	n/a	n/a	483	479	491
ClearEvent		n/a	n/a	n/a	271	267	275
GetEvent		n/a	n/a	n/a	411	411	415
WaitEvent	<default>	n/a	n/a	n/a	1119	1115	1199
	fp	n/a	n/a	n/a	1123	1123	1211
GetAlarmBase		411	423	439	415	419	443
GetAlarm		419	427	447	427	431	451

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		475	483	491	483	483	507
SetAbsAlarm		467	471	487	467	471	495
CancelAlarm		399	403	419	407	407	431
InitCounter		607	607	627	591	603	623
GetCounterValue		387	391	379	383	383	387
osek_tick_alarm	<default>	247	255	251	251	255	255
	KL	51	47	51	47	55	55
osek_incr_counter		23	15	19	19	23	23
GetActiveApplicationMode		15	7	3	11	15	7
StartOS		5275	5275	5275	5567	5275	5571
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	147	147	143	151	151	151
InitCOM		19	19	15	15	19	19
CloseCOM		19	19	15	15	19	19
StartCOM		59	59	47	159	159	159
StopCOM		39	39	31	31	31	43
ReadFlag		n/a	n/a	n/a	35	39	35
ResetFlag		n/a	n/a	n/a	31	35	35
ReceiveMessage		351	351	355	567	587	579
GetMessageResource		n/a	n/a	n/a	699	695	699
ReleaseMessageResource		n/a	n/a	n/a	771	775	775
GetMessageStatus		n/a	n/a	n/a	175	179	171
SendMessage	SW	959	1047	1087	1207	1279	1355
	NS	1031	1123	1163	1263	1343	1411
	KL	651	751	783	875	959	1031
ActivateTaskset	SW	1547	2887	2935	1491	2815	3119
	NS	1603	3063	2995	1543	2807	2979
	KL	1255	2531	2651	1459	2851	2699
	SW2	1547	2883	2943	1495	2823	3119
	NS2	1603	3063	2987	1551	2815	2983
	KL2	1255	2539	2659	1459	2859	2699
ChainTaskset	SWL	2239	3567	3719	2439	3647	3879
	SWH	2499	3975	4111	2715	3899	4031
	NSL	2367	3651	3779	2427	3723	4223
	NSH	2683	3839	3987	2823	4163	4231
GetTasksetRef		363	363	359	359	363	367

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		311	315	307	307	315	303
AssignTaskset		303	295	303	303	303	307
RemoveTaskset		311	307	303	303	311	307
TestSubTaskset		315	315	311	311	307	315
TestEquivalentTaskset		303	307	299	299	311	303
TickSchedule	SW	411	2871	2991	1795	3215	3055
	NS	503	2959	3083	1883	3307	3135
	KL	263	2723	2843	1639	3071	2899
	SW2	411	2871	2987	1795	3187	3039
	NS2	503	2959	3083	1875	3275	3115
	KL2	259	2715	2839	1635	3043	2883
AdvanceSchedule	SW	407	2859	2983	1787	3211	3043
	NS	471	2939	3063	1863	3291	3123
	KL	251	2703	2823	1627	3059	2891
	SW2	407	2863	2975	1779	3187	3027
	NS2	479	2939	3059	1863	3263	3103
	KL2	251	2703	2823	1623	3027	2867
StartSchedule		335	335	323	327	327	331
StopSchedule		275	275	275	279	279	283
GetScheduleStatus		287	287	283	287	287	287
GetScheduleValue		287	287	275	279	279	279
GetScheduleNext		63	63	59	59	63	63
SetScheduleNext		111	111	107	115	111	111
GetArrivalpointDelay		79	79	75	79	79	83
SetArrivalpointDelay		99	99	95	95	99	99
GetArrivalpointTasksetRef		67	67	63	55	67	59
GetArrivalpointNext		67	67	63	63	67	67
SetArrivalpointNext		111	111	107	111	111	115
TestArrivalpointWritable		63	63	59	67	63	71
GetExecutionTime		315	315	311	307	311	315
GetLargestExecutionTime		347	347	343	347	339	347
ResetLargestExecutionTime		335	335	331	331	335	343
GetStackOffset		35	35	31	27	35	35

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	67	67	67	67	67	67
	Cat 2	119	139	139	139	139	139

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	67	67	67	67	67	67
	Cat 2	355	367	379	367	367	367

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	67	67	67	67	67	67
	Cat 2	359	375	379	379	379	379

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.

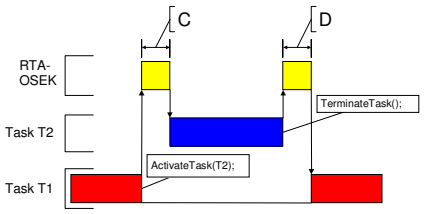

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

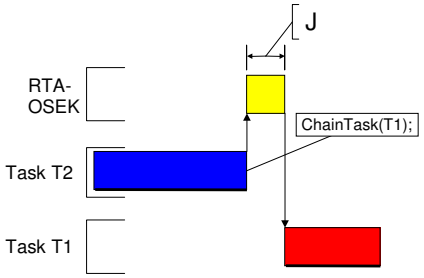

Figure 2: Task Chaining

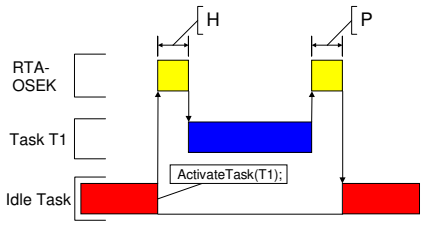

Figure 3: Task Activation from Idle Task

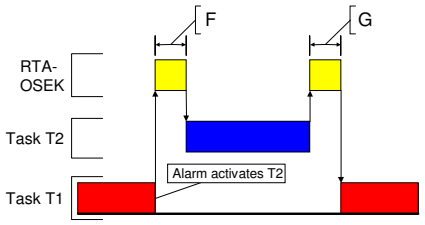

Figure 4: Task Activation from an Alarm

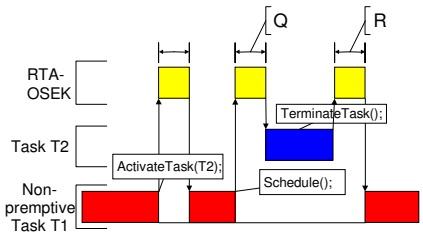

Figure 5: Non-Premptive Task Calls Schedule()

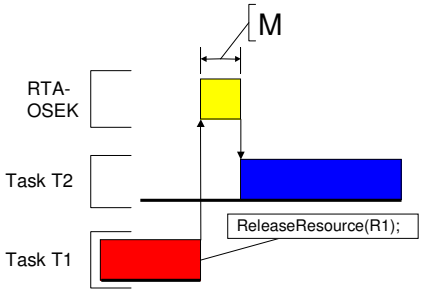

Figure 6: Blocked Task Activated by ReleaseResource()

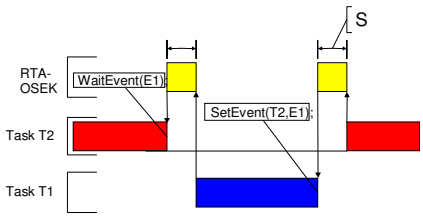

Figure 7: Waiting Task Activated by SetEvent()

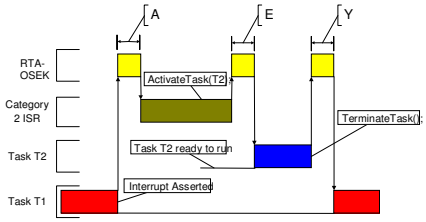

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	179	243	287	183	255	287
Figure 1: D	Heavy, Basic/Extended	235	291	335	323	323	359
ChainTask	Light, Basic	443	547	683	451	567	715
Figure 2: J	Heavy, Basic/Extended	895	1051	1203	999	1095	1283
Pre-emption	Light, Basic	343	451	587	359	463	631
Figure 1: C	Heavy, Basic/Extended	415	519	647	487	563	739
From idle task	Light, Basic	351	451	591	355	459	635
Figure 3: H	Heavy, Basic/Extended	415	519	651	495	563	743
Triggered by alarm	Light, Basic	507	615	739	523	623	787
Figure 4: F	Heavy, Basic/Extended	571	675	803	651	719	895
Schedule	Light, Basic	315	339	443	323	347	439
Figure 5: Q	Heavy, Basic/Extended	383	399	499	455	451	559
Release resource	Light, Basic	315	339	427	323	351	419
Figure 6: M	Heavy, Basic/Extended	379	395	487	459	463	531
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	711	703	943
From category 2 ISR	Light, Basic	327	371	455	363	375	447
Figure 8: E	Heavy, Basic/Extended	395	431	515	499	495	567

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	523	547	587	507	539	579
Figure 1: D	Heavy, Basic/Extended	575	591	627	607	611	651
ChainTask	Light, Basic	795	895	1003	775	875	1047
Figure 2: J	Heavy, Basic/Extended	1615	1699	1839	1667	1719	1931
Pre-emption	Light, Basic	571	663	771	571	659	827
Figure 1: C	Heavy, Basic/Extended	611	711	831	699	759	927
From idle task	Light, Basic	575	639	767	571	643	827
Figure 3: H	Heavy, Basic/Extended	627	723	831	691	763	935
Triggered by alarm	Light, Basic	735	823	939	723	811	995
Figure 4: F	Heavy, Basic/Extended	783	867	995	847	911	1091
Schedule	Light, Basic	535	527	623	531	531	623
Figure 5: Q	Heavy, Basic/Extended	595	595	683	647	655	747
Release resource	Light, Basic	543	527	623	527	543	611
Figure 6: M	Heavy, Basic/Extended	587	595	671	663	659	727
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	899	887	1103
From category 2 ISR	Light, Basic	855	875	943	875	883	951
Figure 8: E	Heavy, Basic/Extended	907	939	1007	991	995	1075

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	743	767	807	743	775	811
Figure 1: D	Heavy, Basic/Extended	783	803	831	839	831	879
ChainTask	Light, Basic	1311	1423	1523	1319	1419	1575
Figure 2: J	Heavy, Basic/Extended	2347	2471	2611	2423	2511	2679
Pre-emption	Light, Basic	947	1039	1155	943	1047	1191
Figure 1: C	Heavy, Basic/Extended	975	1103	1203	1071	1159	1291
From idle task	Light, Basic	943	1039	1159	955	1047	1187
Figure 3: H	Heavy, Basic/Extended	983	1103	1199	1083	1155	1287
Triggered by alarm	Light, Basic	1207	1299	1399	1203	1299	1451
Figure 4: F	Heavy, Basic/Extended	1255	1351	1459	1327	1399	1539
Schedule	Light, Basic	651	659	747	639	659	759
Figure 5: Q	Heavy, Basic/Extended	691	707	803	779	779	863
Release resource	Light, Basic	843	847	919	903	919	987
Figure 6: M	Heavy, Basic/Extended	887	895	983	1031	1031	1119
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1355	1359	1547
From category 2 ISR	Light, Basic	939	975	1047	971	975	1047
Figure 8: E	Heavy, Basic/Extended	983	1011	1091	1079	1083	1151

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

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	148	148	152	148	148	152
BCC1 lightweight, floating-point	152	152	156	152	152	156
BCC1 heavyweight, integer	216	216	220	216	216	220
BCC1 heavyweight, floating-point	216	216	220	216	216	220
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	220	224	n/a	220	224
BCC2 heavyweight, floating-point	n/a	220	224	n/a	220	224
ECC1 heavyweight, integer	n/a	n/a	n/a	256	256	232
ECC1 heavyweight, floating-point	n/a	n/a	n/a	256	256	232
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	148	148	152	148	148	152
BCC1 lightweight, floating-point	152	152	156	152	152	156
BCC1 heavyweight, integer	216	216	220	216	216	220
BCC1 heavyweight, floating-point	216	216	220	216	216	220
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	220	224	n/a	220	224
BCC2 heavyweight, floating-point	n/a	220	224	n/a	220	224
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	240
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	240

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	148	148	152	148	148	152
BCC1 lightweight, floating-point	152	152	156	152	152	156
BCC1 heavyweight, integer	216	216	220	216	216	220
BCC1 heavyweight, floating-point	216	216	220	216	216	220
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	224	228	n/a	224	228
BCC2 heavyweight, floating-point	n/a	224	228	n/a	224	228
ECC1 heavyweight, integer	n/a	n/a	n/a	276	276	244
ECC1 heavyweight, floating-point	n/a	n/a	n/a	276	276	244
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	244
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	244
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	148	148	152	148	148	152
BCC1 lightweight, floating-point	152	152	156	152	152	156
BCC1 heavyweight, integer	216	216	220	216	216	220
BCC1 heavyweight, floating-point	216	216	220	216	216	220
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	224	228	n/a	224	228
BCC2 heavyweight, floating-point	n/a	224	228	n/a	224	228
ECC1 heavyweight, integer	n/a	n/a	n/a	240	240	244
ECC1 heavyweight, floating-point	n/a	n/a	n/a	240	240	244
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	244
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	244

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.