RTA-OSEK

Binding Manual: Blackfin®/ADI

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

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

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

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

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

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: M00098-001

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

Contents

1 About this Guide

This guide provides target-specific information for the Blackfin®/ADI 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.

The Blackfin/ADI port of RTA-OSEK currently supports the following Blackfin variants: ADSP-BF522, ADSP-BF525, ADSP-BF527, ADSP-BF531, ADSP-BF532, ADSP-BF533, ADSP-BF534, ADSP-BF536, ADSP-BF537, ADSP-BF538, ADSP-BF539, ADSP-BF541, ADSP-BF542, ADSP-BF544, ADSP-BF548, ADSP-BF549, and ADSP-BF561.

Note: RTA-OSEK is designed to run on a single core, ADSP-BF561 is a dual core processor, on which RTA-OSEK can be executed on only one of the available cores.

2.1 Compiler

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

Vendor	Analog Devices
Compiler	VisualDSP++™ v5.0 C/C++ Compiler Blackfin
Version	8.0.2.4

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

Option	Description
`-c`	Do not link
`-proc ADSP-xxxxx`	Generate code for Blackfin variant ADSP-xxxxx, where xxxxx is one of the following list: BF522, BF525, BF527, BF531, BF532, BF533, BF534, BF536, BF537, BF538, BF539, BF541, BF542, BF544, BF548, BF549 or BF561

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.

2.1.1 CPU_TYPE environment variable

To support the range of Blackfin CPU types when building an application within the RTA-OSEK environment a DOS environment variable `CPU_TYPE` is used. `CPU_TYPE` should be set to be equal to a CPU type from the following list: ADSP-BF522, ADSP-BF525, ADSP-BF527, ADSP-BF531, ADSP-BF532, ADSP-BF533, ADSP-BF534, ADSP-BF536, ADSP-BF537, ADSP-BF538, ADSP-BF539, ADSP-BF541, ADSP-BF542, ADSP-BF544, ADSP-BF548, ADSP-BF549, and ADSP-BF561. The `-proc` command line option utilizes the `CPU_TYPE` variable when compiling and assembling source files; this is demonstrated in the RTA-OSEK example application.

2.2 Assembler

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

Vendor	Analog Devices
Assembler	VisualDSP++ v5.0 Blackfin Assembler
Version	2.7.2.33

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

Option	Description
`-proc ADSP-xxxxx`	Generate code for Blackfin variant ADSP-xxxxx, where xxxxx is one of the following list: BF522, BF525, BF527, BF531, BF532, BF533, BF534, BF536, BF537, BF538, BF539, BF541, BF542, BF544, BF548, BF549 or BF561

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

2.3 Linker/Locator

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

Option	Description
`-proc ADSP-xxxxx`	Generate code for Blackfin variant ADSP-xxxxx, where xxxxx is one of the following list: BF522, BF525, BF527, BF531, BF532, BF533, BF534, BF536, BF537, BF538, BF539, BF541, BF542, BF544, BF548, BF549 or BF561

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

Sections	ROM/RAM	Description
`os_pid`	ROM	RTA-OSEK read-only data
`os_pird`	ROM	RTA-OSEK initialization data
`os_text`	ROM	RTA-OSEK code

Sections	ROM/RAM	Description
`os_pir`	RAM	RTA-OSEK uninitialized data; initialized during StartOS()
`os_pir2`	RAM	RTA-OSEK initialized data; must be initialized during C-startup
`os_pur`	RAM	RTA-OSEK initialized data; zeroed during StartOS()
`os_pur2`	RAM	RTA-OSEK uninitialized data; must be zeroed during C-startup
`os_trace_ram`	RAM	RTA-TRACE uninitialized data; must be zeroed during C-startup

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

C Library Functions	Description
`setjmp`	VisualDSP++ v5.0 library `setjmp` routine
`longjmp`	VisualDSP++ v5.0 library `longjmp` routine

2.4 Debugger

Information about ORTI for RTA-OSEK can be found in the *RTA-OSEK ORTI Guide*

At the time of writing, we were not aware of any debuggers for the ADI Blackfin with support for ORTI.

If you are using an ORTI version 2.0 aware debugger on this platform you can use the "Unknown ORTI debugger" option in the RTA-OSEK GUI to generate an ORTI output file. The ORTI generated will not have been tested on the debugger and, therefore, is not guaranteed to work.

Please contact LiveDevices if you have any questions about ORTI support in RTA-OSEK.

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for Blackfin®/ADI. 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 *Analog Devices Blackfin Processor Hardware Reference Manual relating to the specific chip variant*.

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

IPL Value	**IMASK Binary**	**Description**
0	`11xx xxxx xxx1 1111`	User level
1	`110x xxxx xxx1 1111`	General Interrupt 13
2	`1100 xxxx xxx1 1111`	General Interrupt 12
3	`1100 0xxx xxx1 1111`	General Interrupt 11
4	`1100 00xx xxx1 1111`	General Interrupt 10
5	`1100 000x xxx1 1111`	General Interrupt 9
6	`1100 0000 xxx1 1111`	General Interrupt 8
7	`1100 0000 0xx1 1111`	General Interrupt 7
8	`1100 0000 00x1 1111`	Core Timer
9	`1100 0000 0001 1111`	Hardware Error
10	`1100 0000 0001 1111`	Exception
11	`1100 0000 0001 1111`	Non-Maskable interrupt
Notes:	x is either 0 or 1 depending on whether the level is used by the application or not respectively RTA-OSEK does not make use of the Blackfin core user mode and IPL 0 does not correspond to this. IPL 0 is the lowest priority (IVG15) in the Blackfin core supervisor mode. IVG14 is reserved for use by RTA-OSEK. The lower 5 bits of IMASK are always set to 1 by the hardware	

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
Vector Offset	Legality
0 and 1	Reset vector is outside of the scope of RTA-OSEK
2 and 3	Category 1 interrupts only
4	CPU Reserved
5 to 13	Category 1 and 2 interrupts
14 and 15	Reserved by RTA-OSEK

The valid base addresses for the vector table are:

Base Address	Notes
0xFFE02000	The entries to the Event Vector Table are entered as core Memory Mapped Registers (MMRs) `EVT0` to `EVT15`, which are initialized in `StartOS()`. There is no scope for changing the base address

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 ADI C compiler can generate appropriate interrupt handling code for a C function decorated with the `EX_REENTRANT_HANDLER()` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

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

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

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

Vector Offset	Label
5 to 13	`os_ist_VV`
eg :	`e.g. os_ist_11`

3.1.6 Operating Mode

The processor is exclusively run in supervisor mode and the user should never enter user mode. Manipulation of `IMASK` is performed by the RTA-OSEK Component and users should not attempt to access it directly.

3.1.7 Manual Vector Table Generation

The Blackfin's vector table is a set of registers that form part of a core peripheral. Therefore, the vectors cannot be initialized in the conventional way of placing values in ROM. Instead, vector addresses are located in MMRs `EVT0` to `EVT15`. When manual vector table generation is enabled, the RTA-OSEK Component calls a function, with prototype `void os_min_vec_init(void)`, which must be implemented by the user. This function is responsible for ensuring vector registers are initialized, including the reserved vector `EVT14` that must be initialized to `os_IST_exit`. The example code below initializes the vector registers for an application with one Category 1 interrupt at vector `EVT8` and one Category 2 interrupt at `EVT11`.

```
#include <stddef.h>
#include <ccblkfn.h>

typedef void (*os_vector)(void);

extern void os_IST_exit(void);
extern void os_ist_11(void);
extern void category_1_ist(void);

void os_min_vec_init(void)
{
  *((os_vector *)EVT5) = NULL;
  *((os_vector *)EVT6) = NULL;
  *((os_vector *)EVT7) = NULL;
  *((os_vector *)EVT8) = category_1_ist;
  *((os_vector *)EVT9) = NULL;
  *((os_vector *)EVT10) = NULL;
  *((os_vector *)EVT11) = os_ist_11;
  *((os_vector *)EVT12) = NULL;
  *((os_vector *)EVT13) = NULL;
  *((os_vector *)EVT14) = os_IST_exit;
}
```

3.1.8 Use of Category 1 interrupts before StartOS()

The EVT registers are initialized by the RTA-OSEK Component during the `StartOS()` API call. If Category 1 Interrupt are used in an application before `StartOS()` then the associated `EVT` registers for these interrupts must be initialized by the user.

3.2 Register Settings

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

Register	**Notes**
`EVT`	Event Vector Table Register

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

Register	**Notes**
`IMASK`	Used to set the OS priority level
`EVT14`	Reserved for use by RTA-OSEK
`EVT15`	RTA-OSEK does not use the Blackfin Core user mode
`RETI`	RTA-OSEK uses the `RETI` register in its API calls. All Category 1 interrupt handlers should be reentrant i.e. use the EX_REENTRANT_HANDLER function modifier not the EX_INTERRUPT_HANDLER function modifier

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): 116

Timing

API max usage (bytes): 128

Extended

API max usage (bytes): 200

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

3.3.3 Stack section

To support ECC tasks RTA-OSEK needs the following Stack symbols (stack bottom and stack top) to be assigned to the symbols defined in linker script file (as demonstrated in the example application linker script Blackfin.ldf).

Stack symbol	Symbol in linker script
_OS_STACK_BOTTOM	ldf_stack_space
_OS_STACK_TOP	ldf_stack_end

e.g.

```
    stack_and_heap_L1_data_a
    {
        INPUT_SECTION_ALIGN(4)
        ldf_stack_space =
        stack_and_heap_in_L1_data_a;
        /* Define a label used by RTA-OSEK to
        mark the bottom of the system stack */
        _OS_STACK_BOTTOM = ldf_stack_space;

        ldf_stack_end = (ldf_stack_space +
        ((stack_and_heap_in_L1_data_a_length *
        STACK_SIZE) / STACKHEAP_SIZE) - 4 ) &
        0xfffffffc;

        /* Define a label used by RTA-OSEK to
        mark the top of the system stack */
        _OS_STACK_TOP = ldf_stack_end;
    } > MEM_L1_DATA_A
```

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	**ADSP-BF537**
Clock speed (MHz)	200
Code memory	Internal SRAM
Read-only data memory	Internal SRAM
Read-write data memory	Internal SRAM

It should be noted that the timing of code execution depends upon a number of factors such as the alignment of code and data objects, stack alignment when an interrupt fires, and the relative location of data objects between two consecutive data accesses. As execution times therefore depend upon configuration the timing information in this section should be used solely as a guideline.

4.1 Functionality

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

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

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 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
Limits for the number of application modes	4294967295					

4.2 Hardware Resources

4.2.1 ROM and RAM Overheads

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	46	46	46	46	46	46
	ROM	186	186	202	246	246	262
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	66	66	66	66	66	66
	ROM	260	260	278	320	320	338
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	84	84	84	84	84	84
	ROM	310	310	328	374	374	392
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

4.2.2 ROM and RAM for OSEK OS Objects

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

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

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	36	36	36	36	36	36
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	40	40	40	40	40	40
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	48	56	n/a	48	56
ECC1, Integer task	RAM	n/a	n/a	n/a	168	168	168
	ROM	n/a	n/a	n/a	60	60	60
ECC1, floating-point task	RAM	n/a	n/a	n/a	170	170	170
	ROM	n/a	n/a	n/a	60	60	60
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	170
	ROM	n/a	n/a	n/a	n/a	n/a	68
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	172
	ROM	n/a	n/a	n/a	n/a	n/a	68
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	76	76	76	76	76	76
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	102	102	102	102	102	102
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	24	24	24	24	24	24
Counter	RAM	4	4	4	4	4	4
	ROM	114	114	114	114	114	114
Message	RAM	11	11	11	31	31	31

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	90	90	90	90	90	90
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	52	52	52	52	52	52
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	60	68	n/a	60	68

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ECC1, Integer task	RAM	n/a	n/a	n/a	180	180	180
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating-point task	RAM	n/a	n/a	n/a	182	182	182
	ROM	n/a	n/a	n/a	72	72	72
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	182
	ROM	n/a	n/a	n/a	n/a	n/a	80
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	184
	ROM	n/a	n/a	n/a	n/a	n/a	80
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	146	146	146	146	146	146
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	154	154	154	154	154	154
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	24	24	24	24	24	24
Counter	RAM	4	4	4	4	4	4
	ROM	114	114	114	114	114	114
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	90	90	90	90	90	90
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	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	
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	16	16	16	16	16	16
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	68	76	n/a	68	76
ECC1, Integer task	RAM	n/a	n/a	n/a	184	184	184
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	186	186	186
	ROM	n/a	n/a	n/a	80	80	80
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	186
	ROM	n/a	n/a	n/a	n/a	n/a	88
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	188
	ROM	n/a	n/a	n/a	n/a	n/a	88
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	158	158	158	158	158	158
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	166	166	166	166	166	166
Resource	RAM	8	8	8	8	8	8

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	28	28	28	28	28	28
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	28	28	28	28	28	28
Counter	RAM	4	4	4	4	4	4
	ROM	118	118	118	118	118	118
Message	RAM	11	11	11	31	31	31
	ROM	24	24	24	60	60	60
Flag	RAM	4	4	4	4	4	4
	ROM	1	1	1	1	1	1
Message resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	90	90	90	90	90	90
ScheduleTable Expiry	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Arrivalpoint (writable)	RAM	20	20	20	20	20	20
	ROM	20	20	20	20	20	20
Schedule	RAM	20	20	20	20	20	20
	ROM	44	44	44	44	44	44
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

4.2.3 Size of Linkable Modules

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

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

The call variants are as follows:

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

Variant	Description
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.
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	174	230	266	182	242	310
	NS		134	192	226	144	204	272
	KL	2	106	166	206	116	178	252
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	32	32	32	32	32	32
ChainTask	SWL	1, 8	170	230	268	180	242	316
	SWH	1, 9	204	264	298	220	276	354
	NSL	8	170	230	268	180	242	316
	NSH	9	192	252	286	208	264	342
Schedule			110	110	134	110	110	134
GetTaskID			36	36	36	36	36	36
GetTaskState			80	80	80	104	104	104
EnableAllInterrupts			22	22	22	22	22	22
DisableAllInterrupts			18	18	18	18	18	18
ResumeAllInterrupts			42	42	42	42	42	42
SuspendAllInterrupts			40	40	40	40	40	40
ResumeOSInterrupts			44	44	44	44	44	44
SuspendOSInterrupts			80	80	80	80	80	80
GetResource	Task	7	26	26	30	26	26	30
	Combined	6	92	92	92	92	92	92
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	94	94	94	94	94	94
	Combined	6	186	186	186	186	186	186

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	154	154	246
	NS		n/a	n/a	n/a	92	92	184
	NS1i	10	n/a	n/a	n/a	44	n/a	n/a
	KL	2	n/a	n/a	n/a	68	68	164
	KL1i	2, 10	n/a	n/a	n/a	26	n/a	n/a
ClearEvent			n/a	n/a	n/a	66	66	66
GetEvent			n/a	n/a	n/a	14	14	14
WaitEvent	<default>		n/a	n/a	n/a	304	304	564
	fp	11	n/a	n/a	n/a	360	360	660
	1i	10	n/a	n/a	n/a	36	n/a	n/a
GetAlarmBase			44	44	44	44	44	44
GetAlarm			92	92	92	92	92	92
SetRelAlarm			382	382	382	382	382	382
SetAbsAlarm			442	442	442	442	442	442
CancelAlarm			80	80	80	80	80	80
InitCounter			46	46	46	46	46	46
GetCounterValue			48	48	48	48	48	48
GetScheduleTableStatus		34	94	132	132	94	132	132
NextScheduleTable		34	118	254	254	118	254	254
StartScheduleTable		34	154	218	216	154	216	218
StopScheduleTable		34	122	160	160	122	160	160
ScheduleTable expiry point	ActivateTask		28	28	28	28	28	28
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	32	32	32
ScheduleTable expiry point	Callback		18	18	18	18	18	18
ScheduleTable expiry point	Tick counter		24	24	24	24	24	24
ScheduleTable expiry point	Final		62	62	62	62	62	62
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	46	46	46	46	46	46
Process container	Non-Yielding	33	24	24	24	24	24	24
osek_tick_alarm	<default>		58	58	58	58	58	58
	KL	2	36	36	36	36	36	36
osek_incr_counter			28	28	28	28	28	28
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			234	234	234	234	234	234
ShutdownOS	NoHook	12	44	44	44	44	44	44
	Hook	13	52	52	52	52	52	52

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
InitCOM			10	10	10	10	10	10
CloseCOM			10	10	10	10	10	10
StartCOM			38	38	38	38	38	38
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	56	56	56	158	158	158
	CCCB	15	158	158	158	158	158	158
GetMessageResource			52	52	52	52	52	52
ReleaseMessageResource			52	52	52	52	52	52
GetMessageStatus			44	44	44	44	44	44
SendMessage	SW CCCA	1, 14	94	94	94	216	216	216
	SW CCCB	1, 15	204	204	204	216	216	216
	NS CCCA	14	94	94	94	216	216	216
	NS CCCB	15	204	204	204	216	216	216
	KL CCCA	2, 14	72	72	72	192	192	192
	KL CCCB	2, 15	180	180	180	192	192	192
main_dispatch	NoHook	12	170	170	214	170	170	214
	Hook	13	220	220	262	220	220	262
sub_dispatch	B1LF	19	34	34	34	34	34	34
	B1HI	20	110	110	110	110	110	110
	B1HF	21	118	118	118	118	118	118
	B2LI	22	n/a	90	114	n/a	90	114
	B2LF	23	n/a	98	122	n/a	98	122
	B2HI	24	n/a	246	310	n/a	246	310
	B2HF	25	n/a	254	318	n/a	254	318
	E1HI	26	n/a	n/a	n/a	352	352	428
	E1HF	27	n/a	n/a	n/a	360	360	436
	E2HI	28	n/a	n/a	n/a	n/a	n/a	428
	E2HF	29	n/a	n/a	n/a	n/a	n/a	436
ErrorHook support		16	54	54	54	54	54	54
	ServiceID	17	68	68	68	68	68	68
	Parameters	18	80	80	80	80	80	80
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	170	276	320	194	316	388

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	NS		136	236	282	154	276	348
	KL	2	104	210	256	124	250	322
ChainTaskset	SWL	1, 8	186	288	334	198	316	380
	SWH	1, 9	224	338	384	236	364	442
	NSL	8	186	288	334	198	316	380
	NSH	9	210	326	372	222	354	430
GetTasksetRef			12	12	12	12	12	12
MergeTaskset			42	42	42	42	42	42
AssignTaskset			12	12	12	12	12	12
RemoveTaskset			46	46	46	46	46	46
TestSubTaskset			52	52	52	52	52	52
TestEquivalentTaskset			50	50	50	50	50	50
TickSchedule	SW	1	240	200	200	200	200	200
	NS		194	140	140	140	140	140
	KL	2	168	116	116	116	116	116
AdvanceSchedule	SW	1	214	186	186	186	186	186
	NS		170	134	134	134	134	134
	KL	2	152	112	112	112	112	112
StartSchedule			84	84	84	84	84	84
StopSchedule			68	68	68	68	68	68
GetScheduleStatus			98	98	98	98	98	98
GetScheduleValue			76	76	76	76	76	76
GetScheduleNext			14	14	14	14	14	14
SetScheduleNext			10	10	10	10	10	10
GetArrivalpointDelay			12	12	12	12	12	12
SetArrivalpointDelay			10	10	10	10	10	10
GetArrivalpointTasksetRef			14	14	14	14	14	14
GetArrivalpointNext			12	12	12	12	12	12
SetArrivalpointNext			10	10	10	10	10	10
TestArrivalpointWritable			36	36	36	36	36	36
GetExecutionTime			10	10	10	10	10	10
GetLargestExecutionTime			10	10	10	10	10	10
ResetLargestExecutionTime			10	10	10	10	10	10
GetStackOffset			18	18	18	18	18	18

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	174	230	266	182	242	310
	NS		134	192	226	144	204	272
	KL	2	106	166	206	116	178	252
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	32	32	32	32	32	32
ChainTask	SWL	1, 8	170	230	268	180	242	316
	SWH	1, 9	204	264	298	220	276	354
	NSL	8	170	230	268	180	242	316
	NSH	9	192	252	286	208	264	342
Schedule			134	134	158	134	134	158
GetTaskID			36	36	36	36	36	36
GetTaskState			80	80	80	104	104	104
EnableAllInterrupts			22	22	22	22	22	22
DisableAllInterrupts			18	18	18	18	18	18
ResumeAllInterrupts			42	42	42	42	42	42
SuspendAllInterrupts			40	40	40	40	40	40
ResumeOSInterrupts			44	44	44	44	44	44
SuspendOSInterrupts			80	80	80	80	80	80
GetResource	Task	7	26	26	30	26	26	30
	Combined	6	92	92	92	92	92	92
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	118	118	118	118	118	118
	Combined	6	226	226	226	226	226	226
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	154	154	246
	NS		n/a	n/a	n/a	92	92	184
	NS1i	10	n/a	n/a	n/a	44	n/a	n/a
	KL	2	n/a	n/a	n/a	68	68	164
	KL1i	2, 10	n/a	n/a	n/a	26	n/a	n/a
ClearEvent			n/a	n/a	n/a	66	66	66
GetEvent			n/a	n/a	n/a	14	14	14
WaitEvent	<default>		n/a	n/a	n/a	382	382	662
	fp	11	n/a	n/a	n/a	438	438	738

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	136	n/a	n/a
GetAlarmBase			44	44	44	44	44	44
GetAlarm			92	92	92	92	92	92
SetRelAlarm			382	382	382	382	382	382
SetAbsAlarm			442	442	442	442	442	442
CancelAlarm			80	80	80	80	80	80
InitCounter			46	46	46	46	46	46
GetCounterValue			48	48	48	48	48	48
GetScheduleTableStatus		34	94	132	132	94	132	132
NextScheduleTable		34	118	254	254	118	254	254
StartScheduleTable		34	154	218	216	154	216	218
StopScheduleTable		34	122	160	160	122	160	160
ScheduleTable expiry point	ActivateTask		28	28	28	28	28	28
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	32	32	32
ScheduleTable expiry point	Callback		18	18	18	18	18	18
ScheduleTable expiry point	Tick counter		24	24	24	24	24	24
ScheduleTable expiry point	Final		62	62	62	62	62	62
GetISRID		4	58	58	58	58	58	58
Process container	Yielding	32	46	46	46	46	46	46
Process container	Non-Yielding	33	24	24	24	24	24	24
osek_tick_alarm	<default>		58	58	58	58	58	58
	KL	2	36	36	36	36	36	36
osek_incr_counter			28	28	28	28	28	28
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			290	290	290	290	290	290
ShutdownOS	NoHook	12	44	44	44	44	44	44
	Hook	13	52	52	52	52	52	52
InitCOM			10	10	10	10	10	10
CloseCOM			10	10	10	10	10	10
StartCOM			38	38	38	38	38	38
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	56	56	56	158	158	158
	CCCB	15	158	158	158	158	158	158
GetMessageResource			52	52	52	52	52	52
ReleaseMessageResource			52	52	52	52	52	52

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			44	44	44	44	44	44
SendMessage	SW CCCA	1, 14	94	94	94	216	216	216
	SW CCCB	1, 15	204	204	204	216	216	216
	NS CCCA	14	94	94	94	216	216	216
	NS CCCB	15	204	204	204	216	216	216
	KL CCCA	2, 14	72	72	72	192	192	192
	KL CCCB	2, 15	180	180	180	192	192	192
main_dispatch	NoHook	12	216	216	264	216	216	264
	Hook	13	264	264	310	264	264	310
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	98	98	98	98	98	98
	B1HF	21	106	106	106	106	106	106
	B2LI	22	n/a	60	84	n/a	60	84
	B2LF	23	n/a	68	92	n/a	68	92
	B2HI	24	n/a	230	292	n/a	230	292
	B2HF	25	n/a	238	300	n/a	238	300
	E1HI	26	n/a	n/a	n/a	378	378	454
	E1HF	27	n/a	n/a	n/a	386	386	462
	E2HI	28	n/a	n/a	n/a	n/a	n/a	454
	E2HF	29	n/a	n/a	n/a	n/a	n/a	462
ErrorHook support		16	54	54	54	54	54	54
	ServiceID	17	68	68	68	68	68	68
	Parameters	18	80	80	80	80	80	80
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	102	102	102	102	102	102
ActivateTaskset	SW	1	170	276	320	194	316	388
	NS		136	236	282	154	276	348
	KL	2	104	210	256	124	250	322
ChainTaskset	SWL	1, 8	186	288	334	198	316	380
	SWH	1, 9	224	338	384	236	364	442
	NSL	8	186	288	334	198	316	380
	NSH	9	210	326	372	222	354	430
GetTasksetRef			12	12	12	12	12	12
MergeTaskset			42	42	42	42	42	42
AssignTaskset			12	12	12	12	12	12
RemoveTaskset			46	46	46	46	46	46

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestSubTaskset			52	52	52	52	52	52
TestEquivalentTaskset			50	50	50	50	50	50
TickSchedule	SW	1	240	200	200	200	200	200
	NS		194	140	140	140	140	140
	KL	2	168	116	116	116	116	116
AdvanceSchedule	SW	1	214	186	186	186	186	186
	NS		170	134	134	134	134	134
	KL	2	152	112	112	112	112	112
StartSchedule			84	84	84	84	84	84
StopSchedule			68	68	68	68	68	68
GetScheduleStatus			98	98	98	98	98	98
GetScheduleValue			76	76	76	76	76	76
GetScheduleNext			14	14	14	14	14	14
SetScheduleNext			10	10	10	10	10	10
GetArrivalpointDelay			12	12	12	12	12	12
SetArrivalpointDelay			10	10	10	10	10	10
GetArrivalpointTasksetRef			14	14	14	14	14	14
GetArrivalpointNext			12	12	12	12	12	12
SetArrivalpointNext			10	10	10	10	10	10
TestArrivalpointWritable			36	36	36	36	36	36
GetExecutionTime			118	118	118	118	118	118
GetLargestExecutionTime			16	16	16	16	16	16
ResetLargestExecutionTime			12	12	12	12	12	12
GetStackOffset			18	18	18	18	18	18

Extended

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	286	338	382	298	350	426
	NS		326	380	424	338	392	468
	KL	2	208	262	304	222	274	348
TerminateTask	LExt	3	154	154	154	154	154	154

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	188	188	188	188	188	188
ChainTask	SWL	1, 8	318	386	420	336	398	470
	SWH	1, 9	356	432	474	370	444	512
	NSL	8	378	448	482	398	460	532
	NSH	9	406	480	522	420	492	560
Schedule			286	286	310	286	286	310
GetTaskID			56	56	56	56	56	56
GetTaskState			254	254	254	268	268	268
EnableAllInterrupts			38	38	38	38	38	38
DisableAllInterrupts			34	34	34	34	34	34
ResumeAllInterrupts			114	114	114	114	114	114
SuspendAllInterrupts			60	60	60	60	60	60
ResumeOSInterrupts			108	108	108	108	108	108
SuspendOSInterrupts			100	100	100	100	100	100
GetResource	Task	7	380	380	338	380	380	338
	Combined	6	384	384	384	384	384	384
	CLEx	3	324	324	324	324	324	324
ReleaseResource	Task	7	354	354	354	354	354	354
	Combined	6	480	480	480	480	480	480
	CLEx	3	308	308	308	308	308	308
SetEvent	SW	1	n/a	n/a	n/a	338	338	442
	NS		n/a	n/a	n/a	370	370	474
	NS1i	10	n/a	n/a	n/a	254	n/a	n/a
	KL	2	n/a	n/a	n/a	266	266	368
	KL1i	2, 10	n/a	n/a	n/a	220	n/a	n/a
ClearEvent			n/a	n/a	n/a	188	188	188
GetEvent			n/a	n/a	n/a	210	210	210
WaitEvent	<default>		n/a	n/a	n/a	568	568	860
	fp	11	n/a	n/a	n/a	592	592	884
	1i	10	n/a	n/a	n/a	286	n/a	n/a
GetAlarmBase			190	190	190	190	190	190
GetAlarm			184	184	184	184	184	184
SetRelAlarm			518	518	518	518	518	518
SetAbsAlarm			566	566	566	566	566	566
CancelAlarm			168	168	168	168	168	168
InitCounter			228	228	228	228	228	228
GetCounterValue			194	194	194	194	194	194

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	116	152	152	116	152	152
NextScheduleTable		34	138	272	272	138	272	272
StartScheduleTable		34	174	236	238	174	236	238
StopScheduleTable		34	144	180	180	144	180	180
ScheduleTable expiry point	ActivateTask		28	28	28	28	28	28
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	32	32	32
ScheduleTable expiry point	Callback		18	18	18	18	18	18
ScheduleTable expiry point	Tick counter		24	24	24	24	24	24
ScheduleTable expiry point	Final		62	62	62	62	62	62
GetISRID		4	78	78	78	78	78	78
Process container	Yielding	32	46	46	46	46	46	46
Process container	Non-Yielding	33	24	24	24	24	24	24
osek_tick_alarm	<default>		116	116	116	116	116	116
	KL	2	36	36	36	36	36	36
osek_incr_counter			28	28	28	28	28	28
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			310	310	310	310	310	310
ShutdownOS	NoHook	12	56	56	56	56	56	56
	Hook	13	64	64	64	64	64	64
InitCOM			10	10	10	10	10	10
CloseCOM			10	10	10	10	10	10
StartCOM			62	62	62	62	62	62
StopCOM			50	50	50	50	50	50
ReadFlag			32	32	32	32	32	32
ResetFlag			38	38	38	38	38	38
ReceiveMessage	CCCA	14	174	174	174	278	278	278
	CCCB	15	278	278	278	278	278	278
GetMessageResource			124	124	124	124	124	124
ReleaseMessageResource			126	126	126	126	126	126
GetMessageStatus			116	116	116	116	116	116
SendMessage	SW CCCA	1, 14	206	206	206	326	326	326
	SW CCCB	1, 15	314	314	314	326	326	326
	NS CCCA	14	206	206	206	326	326	326
	NS CCCB	15	314	314	314	326	326	326
	KL CCCA	2, 14	182	182	182	298	298	298
	KL CCCB	2, 15	286	286	286	298	298	298
main_dispatch	NoHook	12	216	216	264	216	216	264

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
	Hook	13	264	264	310	264	264	310
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	98	98	98	98	98	98
	B1HF	21	106	106	106	106	106	106
	B2LI	22	n/a	60	84	n/a	60	84
	B2LF	23	n/a	68	92	n/a	68	92
	B2HI	24	n/a	230	292	n/a	230	292
	B2HF	25	n/a	238	300	n/a	238	300
	E1HI	26	n/a	n/a	n/a	378	378	454
	E1HF	27	n/a	n/a	n/a	386	386	462
	E2HI	28	n/a	n/a	n/a	n/a	n/a	454
	E2HF	29	n/a	n/a	n/a	n/a	n/a	462
ErrorHook support		16	146	146	146	146	146	146
	ServiceID	17	158	158	158	158	158	158
	Parameters	18	174	174	174	174	174	174
validity_checks		3	34	34	34	34	34	34
Timing_dispatch		4	94	94	94	94	94	94
Timing_termination		4	102	102	102	102	102	102
ActivateTaskset	SW	1	334	408	456	374	456	532
	NS		366	438	482	406	486	562
	KL	2	256	338	384	294	382	462
ChainTaskset	SWL	1, 8	372	476	524	408	504	576
	SWH	1, 9	418	536	580	460	564	640
	NSL	8	418	544	590	492	572	640
	NSH	9	466	590	636	524	616	694
GetTasksetRef			168	168	168	168	168	168
MergeTaskset			308	308	308	308	308	308
AssignTaskset			212	212	212	212	212	212
RemoveTaskset			312	312	312	312	312	312
TestSubTaskset			296	296	296	296	296	296
TestEquivalentTaskset			294	294	294	294	294	294
TickSchedule	SW	1	374	330	330	330	330	330
	NS		404	390	390	390	390	390
	KL	2	292	248	248	248	248	248
AdvanceSchedule	SW	1	376	332	332	332	332	332
	NS		414	396	396	396	396	396
	KL	2	292	258	258	258	258	258

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			264	264	264	264	264	264
StopSchedule			208	208	208	208	208	208
GetScheduleStatus			248	248	248	248	248	248
GetScheduleValue			210	210	210	210	210	210
GetScheduleNext			116	116	116	116	116	116
SetScheduleNext			208	208	208	208	208	208
GetArrivalpointDelay			160	160	160	160	160	160
SetArrivalpointDelay			176	176	176	176	176	176
GetArrivalpointTasksetRef			162	162	162	162	162	162
GetArrivalpointNext			160	160	160	160	160	160
SetArrivalpointNext			204	204	204	204	204	204
TestArrivalpointWritable			166	166	166	166	166	166
GetExecutionTime			178	178	178	178	178	178
GetLargestExecutionTime			140	140	140	140	140	140
ResetLargestExecutionTime			138	138	138	138	138	138
GetStackOffset			18	18	18	18	18	18

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

No timer units are reserved by RTA-OSEK.

RTA-OSEK reserves the general interrupt IVG14. As the Blackfin core user mode is not used IVG15 cannot trigger as application code runs at this level.

4.3 Performance

The collection of performance data for the Blackfin®/ADI port of the RTA-OSEK Component was achieved using a timer running 2 times slower than the CPU clock speed. The figures in this section, therefore, have an uncertainty level of up to 2 CPU cycles. The actual times are between 0 and 2 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	80	108	128	82	114	148
	NS	72	96	116	74	102	136
	KL	52	74	96	54	80	116
TerminateTask	LExt	0	0	0	0	0	0
	H	228	230	242	228	230	242
ChainTask	SWL	390	434	476	428	454	514
	SWH	534	574	616	572	598	662
	NSL	390	434	476	428	454	514
	NSH	528	570	612	566	590	656
Schedule	SW	70	70	84	70	70	84
GetTaskID		22	22	22	22	22	22
GetTaskState		50	50	50	72	72	72
EnableAllInterrupts		44	44	44	46	46	46
DisableAllInterrupts		20	20	20	18	18	18
ResumeAllInterrupts		64	64	64	66	66	66
SuspendAllInterrupts		38	38	38	38	38	38
ResumeOSInterrupts		66	66	66	64	64	64
SuspendOSInterrupts		38	38	38	38	38	38
GetResource	Task	30	30	32	28	28	30
	Combined	98	98	98	98	98	98
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	54	54	54	54	54	54
	Combined	154	154	154	156	156	156
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	78	78	84
	NS	n/a	n/a	n/a	78	78	84
	KL	n/a	n/a	n/a	52	52	60
ClearEvent		n/a	n/a	n/a	100	100	100
GetEvent		n/a	n/a	n/a	24	24	24

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
WaitEvent	<default>	n/a	n/a	n/a	568	570	624
	fp	n/a	n/a	n/a	586	588	646
GetAlarmBase		42	42	42	42	42	42
GetAlarm		56	56	56	56	56	56
SetRelAlarm		94	94	94	94	94	94
SetAbsAlarm		82	82	82	82	82	82
CancelAlarm		46	46	46	46	46	46
InitCounter		50	50	50	48	48	48
GetCounterValue		48	48	48	48	48	48
osek_tick_alarm	<default>	54	54	54	54	54	54
	KL	42	42	42	42	42	42
osek_incr_counter		16	16	16	16	16	16
GetActiveApplicationMode		10	10	10	10	10	10
StartOS		670	670	670	670	670	670
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	76	76	76	76	76	76
InitCOM		14	14	14	14	14	14
CloseCOM		16	16	16	16	16	16
StartCOM		64	64	64	82	82	82
StopCOM		18	18	18	18	18	18
ReadFlag		n/a	n/a	n/a	12	12	12
ResetFlag		n/a	n/a	n/a	10	10	10
ReceiveMessage		46	46	46	136	136	136
GetMessageResource		n/a	n/a	n/a	66	66	66
ReleaseMessageResource		n/a	n/a	n/a	106	106	106
GetMessageStatus		n/a	n/a	n/a	50	50	50
SendMessage	SW	146	174	194	236	268	302
	NS	138	162	182	226	256	290
	KL	108	130	150	194	220	256
ActivateTaskset	SW	64	474	504	70	476	520
	NS	54	462	492	58	464	508
	KL	42	448	478	44	450	494
	SW2	64	474	504	70	476	520
	NS2	54	462	492	58	464	508
	KL2	42	448	478	44	450	494
ChainTaskset	SWL	382	806	864	418	820	890

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
	SWH	522	950	1008	558	964	1036
	NSL	384	806	864	418	820	892
	NSH	520	944	1002	556	960	1032
GetTasksetRef		28	28	28	28	28	28
MergeTaskset		36	36	36	36	36	36
AssignTaskset		22	22	22	22	22	22
RemoveTaskset		38	38	38	38	38	38
TestSubTaskset		40	40	40	40	40	40
TestEquivalentTaskset		40	40	40	40	40	40
TickSchedule	SW	114	560	592	158	570	606
	NS	96	542	574	140	552	588
	KL	82	528	560	126	538	574
	SW2	114	560	592	158	564	606
	NS2	96	542	574	140	546	588
	KL2	82	528	560	126	532	574
AdvanceSchedule	SW	110	540	572	138	550	586
	NS	94	524	556	122	534	570
	KL	78	508	540	106	518	554
	SW2	110	540	572	138	544	586
	NS2	94	524	556	122	528	570
	KL2	78	508	540	106	512	554
StartSchedule		82	82	82	82	82	82
StopSchedule		74	74	74	74	74	74
GetScheduleStatus		56	56	56	56	56	56
GetScheduleValue		78	78	78	78	78	78
GetScheduleNext		24	24	24	24	24	24
SetScheduleNext		26	26	26	26	26	26
GetArrivalpointDelay		20	20	20	20	20	20
SetArrivalpointDelay		24	24	24	24	24	24
GetArrivalpointTasksetRef		20	20	20	20	20	20
GetArrivalpointNext		20	20	20	20	20	20
SetArrivalpointNext		22	22	22	22	22	22
TestArrivalpointWritable		30	30	30	30	30	30
GetExecutionTime		18	18	18	16	16	16
GetLargestExecutionTime		22	22	22	22	22	22
ResetLargestExecutionTime		18	18	18	18	18	18

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetStackOffset		18	18	18	18	18	18

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	80	108	128	82	114	148
	NS	72	96	116	74	102	136
	KL	52	74	96	54	80	116
TerminateTask	LExt	0	0	0	0	0	0
	H	418	422	438	420	420	436
ChainTask	SWL	600	644	688	638	664	732
	SWH	730	770	816	772	796	866
	NSL	600	644	688	638	664	732
	NSH	732	774	820	772	796	868
Schedule	SW	70	70	84	70	70	84
GetTaskID		22	22	22	22	22	22
GetTaskState		50	50	50	72	72	72
EnableAllInterrupts		44	44	44	46	46	46
DisableAllInterrupts		20	20	20	18	18	18
ResumeAllInterrupts		64	64	64	66	66	66
SuspendAllInterrupts		38	38	38	38	38	38
ResumeOSInterrupts		66	66	66	64	64	64
SuspendOSInterrupts		38	38	38	38	38	38
GetResource	Task	30	30	32	28	28	30
	Combined	98	98	98	98	98	98
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	54	54	54	54	54	54
	Combined	158	158	158	160	160	160
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	78	78	84
	NS	n/a	n/a	n/a	78	78	84
	KL	n/a	n/a	n/a	52	52	60

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ClearEvent		n/a	n/a	n/a	100	100	100
GetEvent		n/a	n/a	n/a	24	24	24
WaitEvent	<default>	n/a	n/a	n/a	724	726	798
	fp	n/a	n/a	n/a	752	754	806
GetAlarmBase		42	42	42	42	42	42
GetAlarm		56	56	56	56	56	56
SetRelAlarm		94	94	94	94	94	94
SetAbsAlarm		82	82	82	82	82	82
CancelAlarm		46	46	46	46	46	46
InitCounter		50	50	50	48	48	48
GetCounterValue		48	48	48	48	48	48
osek_tick_alarm	<default>	54	54	54	54	54	54
	KL	42	42	42	42	42	42
osek_incr_counter		16	16	16	16	16	16
GetActiveApplicationMode		10	10	10	10	10	10
StartOS		1560	1564	1564	1564	1564	1560
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	76	76	76	76	76	76
InitCOM		14	14	14	14	14	14
CloseCOM		16	16	16	16	16	16
StartCOM		64	64	64	82	82	82
StopCOM		18	18	18	18	18	18
ReadFlag		n/a	n/a	n/a	12	12	12
ResetFlag		n/a	n/a	n/a	10	10	10
ReceiveMessage		46	46	46	136	136	136
GetMessageResource		n/a	n/a	n/a	66	66	66
ReleaseMessageResource		n/a	n/a	n/a	110	110	110
GetMessageStatus		n/a	n/a	n/a	50	50	50
SendMessage	SW	146	174	194	236	268	302
	NS	138	162	182	226	256	290
	KL	106	128	150	194	220	256
ActivateTaskset	SW	64	474	504	70	476	520
	NS	54	462	492	58	464	508
	KL	42	448	478	44	450	494
	SW2	64	474	504	70	476	520
	NS2	54	462	492	58	464	508

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	KL2	42	448	478	44	450	494
ChainTaskset	SWL	592	1016	1076	628	1030	1108
	SWH	718	1146	1208	758	1162	1242
	NSL	592	1016	1076	630	1030	1110
	NSH	716	1150	1210	756	1166	1244
GetTasksetRef		28	28	28	28	28	28
MergeTaskset		36	36	36	36	36	36
AssignTaskset		22	22	22	22	22	22
RemoveTaskset		38	38	38	38	38	38
TestSubTaskset		40	40	40	40	40	40
TestEquivalentTaskset		40	40	40	40	40	40
TickSchedule	SW	114	560	592	158	570	606
	NS	96	542	574	140	552	588
	KL	82	528	560	126	538	574
	SW2	114	560	592	158	564	606
	NS2	96	542	574	140	546	588
	KL2	82	528	560	126	532	574
AdvanceSchedule	SW	110	540	572	138	550	586
	NS	94	524	556	122	534	570
	KL	78	508	540	106	518	554
	SW2	110	540	572	138	544	586
	NS2	94	524	556	122	528	570
	KL2	78	508	540	106	512	554
StartSchedule		82	82	82	82	82	82
StopSchedule		74	74	74	74	74	74
GetScheduleStatus		56	56	56	56	56	56
GetScheduleValue		78	78	78	78	78	78
GetScheduleNext		24	24	24	24	24	24
SetScheduleNext		26	26	26	26	26	26
GetArrivalpointDelay		20	20	20	20	20	20
SetArrivalpointDelay		24	24	24	24	24	24
GetArrivalpointTasksetRef		20	20	20	20	20	20
GetArrivalpointNext		20	20	20	20	20	20
SetArrivalpointNext		22	22	22	22	22	22
TestArrivalpointWritable		30	30	30	30	30	30
GetExecutionTime		92	92	92	92	90	92

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetLargestExecutionTime		30	30	30	30	30	30
ResetLargestExecutionTime		28	28	28	28	28	28
GetStackOffset		18	18	18	18	18	18

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	274	298	316	280	304	338
	NS	300	324	342	306	330	364
	KL	194	218	238	198	224	260
TerminateTask	LExt	506	506	522	506	506	522
	H	596	600	614	600	598	614
ChainTask	SWL	842	894	940	886	918	986
	SWH	970	1022	1074	1012	1046	1110
	NSL	872	926	970	914	950	1016
	NSH	992	1046	1098	1036	1070	1134
Schedule	SW	172	172	186	172	172	186
GetTaskID		30	30	30	30	30	30
GetTaskState		278	278	278	296	296	296
EnableAllInterrupts		50	50	50	52	52	52
DisableAllInterrupts		26	26	26	24	24	24
ResumeAllInterrupts		80	80	80	82	82	82
SuspendAllInterrupts		44	44	44	44	44	44
ResumeOSInterrupts		82	82	82	80	80	82
SuspendOSInterrupts		44	44	44	44	44	44
GetResource	Task	430	430	270	446	446	286
	Combined	264	264	264	282	282	282
	CLEx	284	284	284	302	302	302
ReleaseResource	Task	256	256	256	274	274	274
	Combined	316	316	316	334	334	334
	CLEx	256	256	256	274	274	274
SetEvent	SW	n/a	n/a	n/a	306	306	306

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NS	n/a	n/a	n/a	320	320	320
	KL	n/a	n/a	n/a	236	236	238
ClearEvent		n/a	n/a	n/a	154	154	154
GetEvent		n/a	n/a	n/a	196	196	196
WaitEvent	<default>	n/a	n/a	n/a	912	914	978
	fp	n/a	n/a	n/a	924	924	974
GetAlarmBase		216	216	222	214	214	220
GetAlarm		224	224	230	222	222	228
SetRelAlarm		276	276	282	276	276	282
SetAbsAlarm		256	256	262	256	256	262
CancelAlarm		208	208	214	208	208	214
InitCounter		316	316	328	316	316	326
GetCounterValue		214	214	214	216	216	216
osek_tick_alarm	<default>	134	134	134	134	134	134
	KL	42	42	42	42	42	42
osek_incr_counter		16	16	16	16	16	16
GetActiveApplicationMode		10	10	10	10	10	10
StartOS		1610	1610	1610	1606	1606	1610
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	80	80	80	78	78	80
InitCOM		14	14	14	14	14	14
CloseCOM		16	16	16	16	16	16
StartCOM		76	76	76	96	96	96
StopCOM		30	30	30	30	30	30
ReadFlag		n/a	n/a	n/a	28	28	28
ResetFlag		n/a	n/a	n/a	24	24	24
ReceiveMessage		202	202	202	288	288	288
GetMessageResource		n/a	n/a	n/a	418	418	418
ReleaseMessageResource		n/a	n/a	n/a	398	398	398
GetMessageStatus		n/a	n/a	n/a	122	122	122
SendMessage	SW	510	534	552	592	618	650
	NS	536	560	578	618	644	676
	KL	356	380	402	434	460	498
ActivateTaskset	SW	328	742	774	344	748	796
	NS	358	782	812	372	788	834
	KL	256	672	704	278	676	726

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	SW2	328	742	774	344	748	796
	NS2	358	782	812	372	788	834
	KL2	256	672	704	278	676	726
ChainTaskset	SWL	926	1356	1418	976	1374	1466
	SWH	1044	1490	1550	1094	1504	1594
	NSL	948	1408	1470	1018	1422	1514
	NSH	1074	1536	1596	1128	1554	1638
GetTasksetRef		178	178	178	178	178	178
MergeTaskset		184	184	184	184	184	184
AssignTaskset		88	88	88	88	88	88
RemoveTaskset		186	186	186	186	186	186
TestSubTaskset		184	184	184	184	184	184
TestEquivalentTaskset		184	184	184	184	184	184
TickSchedule	SW	226	858	888	462	870	912
	NS	246	878	908	482	890	932
	KL	148	782	814	388	796	838
	SW2	226	858	888	462	862	910
	NS2	246	878	908	482	882	930
	KL2	148	782	814	388	786	836
AdvanceSchedule	SW	222	852	884	458	866	908
	NS	246	878	910	484	892	934
	KL	158	792	824	396	804	848
	SW2	222	852	884	458	856	906
	NS2	246	878	910	484	882	932
	KL2	158	792	824	396	794	846
StartSchedule		190	190	190	190	190	190
StopSchedule		174	174	174	176	176	176
GetScheduleStatus		164	164	164	162	162	162
GetScheduleValue		168	168	168	168	168	168
GetScheduleNext		48	48	48	48	48	48
SetScheduleNext		100	100	100	100	100	100
GetArrivalpointDelay		68	68	68	68	68	68
SetArrivalpointDelay		76	76	76	76	76	76
GetArrivalpointTasksetRef		58	58	58	58	58	58
GetArrivalpointNext		60	60	60	60	60	60
SetArrivalpointNext		108	108	108	110	110	110

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
TestArrivalpointWritable		66	66	66	66	66	66
GetExecutionTime		168	170	168	170	168	170
GetLargestExecutionTime		164	164	164	164	164	164
ResetLargestExecutionTime		158	158	158	158	158	158
GetStackOffset		18	18	18	18	18	18

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	76	76	76	76	76	76
	Cat 2	146	168	168	168	168	168

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	76	76	76	76	76	76
	Cat 2	294	310	310	310	310	310

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	76	76	76	76	76	76
	Cat 2	294	310	310	310	310	310

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.

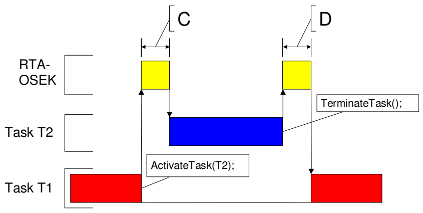

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

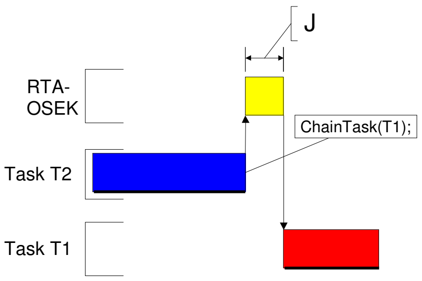

Figure 2: Task Chaining

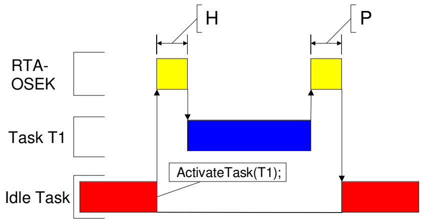

Figure 3: Task Activation from Idle Task

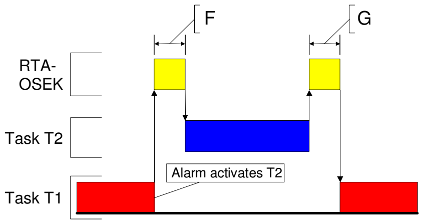

Figure 4: Task Activation from an Alarm

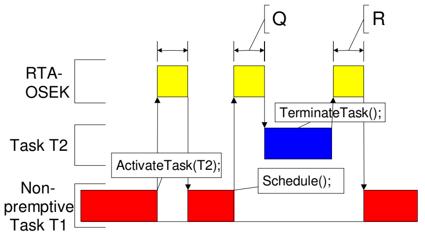

Figure 5: Non-Premptive Task Calls Schedule()

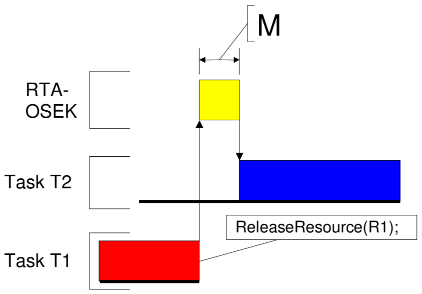

Figure 6: Blocked Task Activated by ReleaseResource()

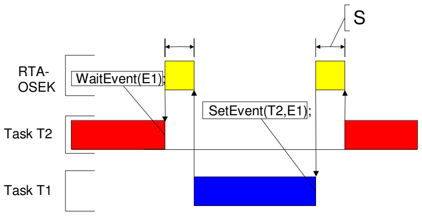

Figure 7: Waiting Task Activated by SetEvent()

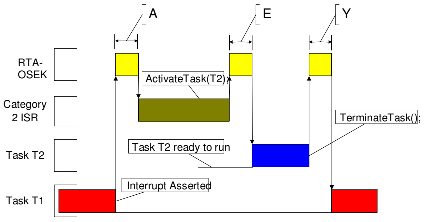

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	104	170	196	104	170	196
Figure 1: D	Heavy, Basic/Extended	230	288	312	288	290	316
ChainTask	Light, Basic	244	290	332	246	290	350
Figure 2: J	Heavy, Basic/Extended	662	764	832	722	766	856
Pre-emption	Light, Basic	188	234	290	190	234	304
Figure 1: C	Heavy, Basic/Extended	340	384	440	378	408	478
From idle task	Light, Basic	192	238	294	194	238	308
Figure 3: H	Heavy, Basic/Extended	344	388	444	382	412	482
Triggered by alarm	Light, Basic	258	304	360	260	304	374
Figure 4: F	Heavy, Basic/Extended	412	456	512	450	480	550
Schedule	Light, Basic	162	182	234	162	182	234
Figure 5: Q	Heavy, Basic/Extended	314	332	384	350	350	402
Release resource	Light, Basic	176	196	234	176	196	234
Figure 6: M	Heavy, Basic/Extended	328	346	384	364	364	402
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	572	572	674
From category 2 ISR	Light, Basic	202	248	286	226	248	286
Figure 8: E	Heavy, Basic/Extended	356	398	436	414	414	452

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	306	340	372	306	338	370
Figure 1: D	Heavy, Basic/Extended	418	446	474	446	446	478
ChainTask	Light, Basic	462	496	542	464	494	562
Figure 2: J	Heavy, Basic/Extended	1052	1110	1182	1088	1116	1224
Pre-emption	Light, Basic	308	342	410	310	342	422
Figure 1: C	Heavy, Basic/Extended	452	496	562	492	520	602
From idle task	Light, Basic	312	346	414	314	346	426

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Figure 3: H	Heavy, Basic/Extended	456	500	566	496	524	606
Triggered by alarm	Light, Basic	378	412	480	380	412	492
Figure 4: F	Heavy, Basic/Extended	524	568	634	564	592	674
Schedule	Light, Basic	282	290	354	282	290	352
Figure 5: Q	Heavy, Basic/Extended	426	444	506	464	462	526
Release resource	Light, Basic	300	308	358	300	308	356
Figure 6: M	Heavy, Basic/Extended	444	462	510	482	480	530
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	642	642	756
From category 2 ISR	Light, Basic	548	574	622	566	572	622
Figure 8: E	Heavy, Basic/Extended	686	720	768	740	738	788

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	502	536	566	504	536	566
Figure 1: D	Heavy, Basic/Extended	598	624	652	626	626	658
ChainTask	Light, Basic	704	746	794	712	746	812
Figure 2: J	Heavy, Basic/Extended	1472	1536	1618	1502	1546	1640
Pre-emption	Light, Basic	466	498	566	472	498	582
Figure 1: C	Heavy, Basic/Extended	612	654	718	652	678	760
From idle task	Light, Basic	470	502	570	476	502	586
Figure 3: H	Heavy, Basic/Extended	616	658	722	656	682	764
Triggered by alarm	Light, Basic	616	648	716	622	648	732
Figure 4: F	Heavy, Basic/Extended	764	806	870	804	830	912
Schedule	Light, Basic	352	360	424	352	360	424
Figure 5: Q	Heavy, Basic/Extended	498	516	576	532	534	596
Release resource	Light, Basic	458	466	516	476	484	534
Figure 6: M	Heavy, Basic/Extended	604	622	668	656	658	706
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	860	862	974
From category 2 ISR	Light, Basic	626	650	700	642	650	700
Figure 8: E	Heavy, Basic/Extended	764	800	844	816	816	866

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		212	212	236	212	212	236
BCC1 lightweight, floating-point		232	232	256	232	232	256
BCC1 heavyweight, integer		392	392	416	392	392	416
BCC1 heavyweight, floating-point		392	392	416	392	392	416
BCC2 lightweight, integer		n/a	240	264	n/a	240	264
BCC2 lightweight, floating-point		n/a	240	264	n/a	240	264
BCC2 heavyweight, integer		n/a	396	420	n/a	396	420
BCC2 heavyweight, floating-point		n/a	396	420	n/a	396	420
ECC1 heavyweight, integer		n/a	n/a	n/a	496	496	520
ECC1 heavyweight, floating-point		n/a	n/a	n/a	496	496	520
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	528
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	528
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		232	232	236	232	232	236
BCC1 lightweight, floating-point		252	252	256	252	252	256
BCC1 heavyweight, integer		412	412	416	412	412	416
BCC1 heavyweight, floating-point		412	412	416	412	412	416
BCC2 lightweight, integer		n/a	260	264	n/a	260	264

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC2 lightweight, floating-point		n/a	260	264	n/a	260	264
BCC2 heavyweight, integer		n/a	416	420	n/a	416	420
BCC2 heavyweight, floating-point		n/a	416	420	n/a	416	420
ECC1 heavyweight, integer		n/a	n/a	n/a	516	516	520
ECC1 heavyweight, floating-point		n/a	n/a	n/a	516	516	520
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	528
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	528

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		256	256	280	256	256	280
BCC1 lightweight, floating-point		276	276	300	276	276	300
BCC1 heavyweight, integer		436	436	460	436	436	460
BCC1 heavyweight, floating-point		436	436	460	436	436	460
BCC2 lightweight, integer		n/a	276	300	n/a	276	300
BCC2 lightweight, floating-point		n/a	276	300	n/a	276	300
BCC2 heavyweight, integer		n/a	440	460	n/a	440	460
BCC2 heavyweight, floating-point		n/a	440	460	n/a	440	460
ECC1 heavyweight, integer		n/a	n/a	n/a	548	548	572
ECC1 heavyweight, floating-point		n/a	n/a	n/a	548	548	572
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	580
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	580
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		276	276	280	276	276	280
BCC1 lightweight, floating-point		296	296	300	296	296	300
BCC1 heavyweight, integer		456	456	460	456	456	460
BCC1 heavyweight, floating-point		456	456	460	456	456	460
BCC2 lightweight, integer		n/a	296	300	n/a	296	300

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
BCC2 lightweight, floating-point	n/a	296	300	n/a	296	300
BCC2 heavyweight, integer	n/a	460	460	n/a	460	460
BCC2 heavyweight, floating-point	n/a	460	460	n/a	460	460
ECC1 heavyweight, integer	n/a	n/a	n/a	568	568	572
ECC1 heavyweight, floating-point	n/a	n/a	n/a	568	568	572
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	580
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	580

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	256	256	280	256	256	280
BCC1 lightweight, floating-point	276	276	300	276	276	300
BCC1 heavyweight, integer	436	436	460	436	436	460
BCC1 heavyweight, floating-point	436	436	460	436	436	460
BCC2 lightweight, integer	n/a	276	300	n/a	276	300
BCC2 lightweight, floating-point	n/a	276	300	n/a	276	300
BCC2 heavyweight, integer	n/a	440	460	n/a	440	460
BCC2 heavyweight, floating-point	n/a	440	460	n/a	440	460
ECC1 heavyweight, integer	n/a	n/a	n/a	572	572	596
ECC1 heavyweight, floating-point	n/a	n/a	n/a	572	572	596
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	604
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	604
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	276	276	280	276	276	280
BCC1 lightweight, floating-point	296	296	300	296	296	300
BCC1 heavyweight, integer	456	456	460	456	456	460
BCC1 heavyweight, floating-point	456	456	460	456	456	460
BCC2 lightweight, integer	n/a	296	300	n/a	296	300

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC2 lightweight, floating-point		n/a	296	300	n/a	296	300
BCC2 heavyweight, integer		n/a	460	460	n/a	460	460
BCC2 heavyweight, floating-point		n/a	460	460	n/a	460	460
ECC1 heavyweight, integer		n/a	n/a	n/a	592	592	596
ECC1 heavyweight, floating-point		n/a	n/a	n/a	592	592	596
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	604
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	604

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 Stack Labels

The linker stack labels used by RTA-OSEK have changed in Visual DSP++ V5.0. The table below shows the stack symbol changes from V3.x to V5.0.

Visual DSP++ V3.x	Visual DSP++ V5.0
ldf_sysstack_space	`_OS_STACK_BOTTOM`
ldf_sysstack_end	`_OS_STACK_TOP`

To convert the stack labels used in an existing v3.x application to those required by v5.0 please refer to section 3.3.

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.