RTA-OSEK

Binding Manual: MPC55xxVLE/WindRiver

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

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

www.etas.de

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

Japan

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

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

www.etas.co.jp

France

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

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

www.etas.fr

Korea

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

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

www.etas.co.kr

Great Britain

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

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

www.etas-uk.net

Copyright Notice

© 2001 - 2006 LiveDevices Ltd. All rights reserved.

Version: RM00072-002

No part of this document may be reproduced without the prior written consent of LiveDevices Ltd. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

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 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 port specific information for the MPC55xxVLE/WindRiver implementation of LiveDevices' RTA-OSEK.

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?

It is assumed that you are a developer. 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.

In this guide you'll see that program code, header file names, C type names, C functions 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

In this chapter, you'll see the important details that you need to know about RTA-OSEK and your toolchain. A port of the RTA-OSEK Component is specific to both the target hardware *and* the compiler toolchain. You must make sure that you build your application with this toolchain.

If you are interested in using a different version of the same toolchain, you should contact LiveDevices to confirm whether or not this is possible.

The MPC55xxVLE/WindRiver supports the single flat memory model supported by the Wind River Systems, Inc. (Diab) toolchain. This toolchain supports the Embedded Application Binary Interface, EABI.

2.1 Compiler

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

Vendor	Wind River Systems, Inc.
Compiler	Wind Power (Diab) C/C++ Compiler for PowerPC
Version	5.3.2.0

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

Option	Description
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.

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
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.
`-g0`	Turn debug mode off.

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

Option	Description
`-Xsmall-data=0`	Place no static or global variables in SDATA
`-Xsmall-const=0`	Place no static or global variables in SCONST
`-Xaddr-data=0x20`	Default to SDA addressing

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

Option	Description
`-g`	Debugging must be disabled.

To support the use of multiple CPU configurations the environment variable CPU_TYPE should be set up to match the desired CPU target (e.g. PPC5534ES:simple).

2.2 Assembler

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

Vendor	Wind River Systems, Inc.
Assembler	Wind Power (Diab) Assembler for the PowerPC
Version	5.3.2.0

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

Option	Description
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.

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

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

Option	Description
`-t%CPU_TYPE%`	Selects the correct target for code generation etc.

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. This section is typically empty in this release. For performance reasons, it can be mapped into RAM (if initialized correctly).
`os_pidf`	ROM	RTA-OSEK read-only data. This section contains RTA-OSEK constant data, all of which is far-addressed (OS_CONST_VAR and OS_CONST_ROM). For performance reasons, it can be mapped into 'far' RAM (if initialized correctly).
`os_pird`	ROM	RTA-OSEK initialization data. This is only accessed during StartOS(). It will normally be located in slow ROM.
`os_pnird`	ROM	RTA-OSEK near initialization data. This is only accessed during StartOS(). It will normally be located in slow ROM.
`os_intvec`	ROM	Vector table (if generated by RTA-OSEK). Should be aligned on a 64KByte boundary
`os_text`	ROM	RTA-OSEK code section.
`os_pir`	RAM	RTA-OSEK initialized data. Must be initialized during C-startup. Can be located in 'far' RAM.
`os_pnir`	RAM	RTA-OSEK near initialized data. Must be initialized during C-startup. Must be placed in the compiler SDA (near addressing)
`os_cntr`	RAM	RTA-OSEK near initialized data. Must be zeroed during C-startup. Must be placed in the compiler SDA (near addressing)
`os_pur`	RAM	RTA-OSEK uninitialized data. Must be zeroed during C-startup.
`os_trace_ram`	RAM	RTA-TRACE buffer. RTA-TRACE buffer. Can be located in 'far' RAM. Does not need to be initialized.

2.4 Debugger

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

ORTI compatible debuggers	Lauterbach TRACE32

The RTA-OSEK GUI outputs a file with the extension `.ort`. This file should be loaded into the debugger with the command `Task.ORTI <file>`. Note that this must be loaded after the executable (`.elf`) file. Please refer to the debugger documentation for further details on its support for ORTI.

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model. 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 *MPC5567 Reference Manual and the e200z6 Core Supplementary Reference Manual*.

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

IPL Value	INTC_CPR Value	MSR[EE] Bit	Description
0	0	1	User level
1-15	1-15	1	INTC Category 1 and 2 interrupts
16	15	0	CPU Category 1 interrupts only

3.1.2 Interrupt Vectors

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

Vector	Legality
0x1 to 0x22	The CPU vectors only handle Category 1 ISRs
0x10000 to the maximum INTC vector for the chip variant in 0x10 steps	The INTC vectors can handle Category 1 and 2 ISRs

The valid base addresses for the vector table are:

Base Address	Notes
IVPR	The base address of the vector table should be aligned to a 64 Kbyte boundary.

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 Wind River Systems, 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

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

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 (VVVVV represents the 5 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
`0xVVVVV`	`os_wrapper_VVVVV`
eg : 0x10330	`os_wrapper_10330`

3.1.6 Processor Mode

RTA-OSEK operates in the processor's supervisor mode and also expects applications to do so.

Important: The application should not set the "Problem State" bit of the Machine State Register, MSR[PR].

3.1.7 INTC vector mode

To reduce the entry time into Category 1 and 2 ISRs it is recommended that the Interrupt Controller is configured in Hardware (HW) vector mode. This section includes notes and assembly language code fragments to assist the use of RTA-OSEK in Software (SW) vector mode. Note that the performance figures have been collected for a system using the HW vector mode.

Provide the Interrupt handler for the SW mode:

When the INTC interrupt controller is operated in SW vector mode a user provided common interrupt exception handler is used to determine the vector of the interrupt request source. When an interrupt is triggered the address of the relevant interrupt handler address is held within the INTC_IACKR register. The Assembly language routine `interrupt_exception_handler` demonstrates a method of reading the INTC_IACKR register and branching to that interrupt handler:

```
interrupt_exception_handler:

; Code to save SRR0 and SRR1

 stwu  r1,-16(r1)       ; Allocate some stack
 stw   r3,8(r1)         ; Store r3
 mfspr r3,ctr           ; Store the CTR register
 stw   r3,12(r1)

 addis r3,0,%hi(0xFFF48010)  ; Form INTC_IACKR
address
 ori   r3,r3,%lo(0xFFF48010)
 lwz   r3,0x0(r3)       ; Load INTC_IACKR, which
clears request to processor
 lwz   r3,0x0(r3)       ; Load address of ISR from
vector table

 ; Code to enable processor recognition of
interrupts and save context required by EABI
 mtspr ctr,r3           ; Move INTC_IACKR contents
into CTR register
                        ; Set up function addr for
the indirect branch
 bcctr 20,0             ; Indirect branch to ISR
function
```

The default operation of RTA-OSEK uses HW vector mode to support interrupt recognition. To operate in SW vector mode the RTA-OSEK library function `os_mid_wrapper()` should be replaced with the following locally provided version. In HW vector mode the function `os_mid_wrapper()` stores and restore common interrupt context and returns from the interrupt. The function `os_mid_wrapper()` is called after the stack frame of 80 bytes has been reserved and the R3 register has been loaded with an ISR specific address. In SW vector mode additional context restoration needs to be performed to restore the context used by the common interrupt exception handler; this is added to `os_mid_wrapper()`. In the following example the context restoration instructions appear after the label `interrupt_exception_end()`. The method of calling the interrupt handling routine differs in applications built in the RTA-OSEK standard build to all others. If the application uses the standard build then the following example should be built with the preprocessor macro `OS_STANDARD_BUILD` defined.

```
os_mid_wrapper:
  stw   r0,8(r1)            ; Save interrupt context
following the EABI

  mfspr r0,srr0             ; Save the SRR0/1 to allow
nested interrupts
  stw   r0,12(r1)
  mfspr r0,srr1
  stw   r0,16(r1)

  mtmsr r0                  ; Restore pre-interrupted
msr
                            ; (i.e. set EE bit and SPE
bit if enabled)

                            ; Cat 1 blocking ends here

  mfspr r0,ctr                    ; Save the non GPR
regs
  stw   r0,20(r1)
  mfspr r0,xer
  stw   r0,24(r1)
  mfcr  r0
  stw   r0,28(r1)
  mfspr r0,lr
  stw   r0,32(r1)

  .ifdef OS_STANDARD_BUILD
  mtspr ctr,r3                    ; Set up function addr
for the indirect branch
  .endif ; OS_STANDARD_BUILD

  stw   r4,44(r1)
  stw   r5,48(r1)
  stw   r6,52(r1)
```

```
 stw   r7,56(r1)
 stw   r8,60(r1)
 stw   r9,64(r1)
 stw   r10,68(r1)
 stw   r11,72(r1)
 stw   r12,76(r1)

 .ifdef OS_STANDARD_BUILD
 bcctrl      20,0                ; Indirect branch to
ISR function
 .endif ; OS_STANDARD_BUILD

 bl    os_wrapper        ; Call inner wrapper

                         ; Restore the context
 lwz   r0,32(r1)         ; Restore the LR with a load
inserted
 lwz   r12,76(r1)
 mtspr lr,r0

 lwz   r11,72(r1)
 lwz   r10,68(r1)
 lwz   r9,64(r1)
 lwz   r8,60(r1)
 lwz   r7,56(r1)
 lwz   r6,52(r1)

 lwz   r0,28(r1)         ; Restore the CRF with a
load inserted
 lwz   r5,48(r1)
 mtcrf 0xff,r0
 lwz   r0,24(r1)
 mtspr xer,r0
 lwz   r0,20(r1)
 mtspr ctr,r0

                         ; Cat 1 blocking starts here
 wrteei     0            ; Clear EE bit

                         ; Return the IPL level to
that before the interrupt triggered
                         ; INTC_CPR is at address
0xFFF48008
 addis r4,0,%hiadj(0xFFF48008)
 stw   r3,%lo(0xFFF48008)(r4)

 lwz   r4,44(r1)
 lwz   r3,40(r1)         ; Restore the remaining
context

 lwz   r0,16(r1)         ; Restore SRR0/1
 mtspr srr1,r0
 lwz   r0,12(r1)
```

```
 mtspr srr0,r0
 lwz   r0,8(r1)

 addi  r1,r1,80          ; Restore the SP

interrupt_exception_end:
 ; Code to restore context required by the SW mode
vector handler
 ; Code to restore SRR0 and SRR1

 lwz   r3,12(r1)         ; Restore the ctr
 mtspr ctr,r3
 lwz   r3,8(r1)          ; Restore r3
 addi  r1,r1,16          ; Restore the SP

 rfi                     ; Return from the interrupt
                         : Cat 1 blocking ends
```

Generating a vector table and initializing the INTC_IACKR register

The vector table used by the interrupt controller in SW vector mode is not compatible with the vector table generated by RTA-OSEK for use in HW vector mode. The example handler routine `interrupt_exception_handler` expects that the vector table consists of 4-byte addresses of the interrupt handler functions. In this case the user should generate a vector table manually as described in section 3.1.5. The table should be aligned to a 2-Kbyte boundary and the address of the start of the table should be loaded into the INTC_IACKR register.

Initializing the IVOR4 registers:

The SW vector mode common interrupt exception handler's location is determined by an address derived from special purpose registers IVPR and IVOR4. The IVOR4 register should hold the lower 2-bytes of the address of the SW vector mode common interrupt exception handler offset form the value of the IVPR.

3.1.8 INTC and CPU Vector Offset Mapping

The MPC55XX has two exception sources the CPU and the Interrupt Controller.

CPU interrupts and exceptions: The addresses of the handler functions for the CPU exceptions are formed by combining the contents of the IVPR register and the IVORx register specific to the exception source. The CPU exception sources are characterized in an integer array `os_CPU_vectors`, which contains the addresses of the interrupt handlers for the ISRs. The offset addresses, shown in Section 3.1.2, illustrate the direct mapping between the range of entries and the interrupt sources that should be used in RTA-OSEK. The array can be used to initialize the IVPR and IVORx registers; this is demonstrated in the example application. As the top 16 bits of the address are common to all CPU interrupt handlers they must reside in a common 64 Kbytes block of memory.

INTC interrupts and exceptions: In hardware vector mode the address of the handler function for an INTC interrupt is formed by combining the contents of the IVPR register with the offset corresponding to the interrupt source that has triggered. The integer array `os_INTC_vectors` is generated by RTA-OSEK containing the quad word aligned interrupt exception handler addresses for each source. This array should be aligned to a 64 Kbytes boundary with the IVPR containing the top 16 bits of the address of the start of the array; this is demonstrated in the example application. The 64 Kbytes block of memory should contain both the INTC vectors and the CPU interrupt handlers. The offset addresses, shown in Section 3.1.2, illustrate the direct mapping between the range of entries and the interrupt sources that should be used in RTA-OSEK. As the IVPR supplies the top 16 bits of the vector address, the vector offsets supplied to the RTA-OSEK GUI uses bit value 0x10000 to distinguish between CPU and INTC vectors; this bit is masked off when creating a physical vector offset.

3.1.9 Number of supported INTC vectors

The number of vectors available depends upon the PowerPC chip variant selected in RTA-OSEK. Currently the variants directly supported are the MPC5534, the MPC5565, the MPC5567 and the MPC55xx VLE Generic. Further variants can be supported by contacting LiveDevices.

When RTA-OSEK generates an interrupt vector table for the MPC55xx VLE, it only emits data for addresses 0x10000 up to the highest declared interrupt. This allows RTA-OSEK to cope efficiently with chip variants with differently sized vector tables.

3.1.10 INTC PSR register initialization

To assist the user with the initialization of the INTC priority select registers (INTC_PSRs) RTA-OSEK generates the array `os_intc_psr_init` in the file `osgen.s`. The array contains the interrupt priority level for the range of interrupts INTC declared in the application (from 0x10000 to the highest declared interrupt). The array is terminated by a byte value 0xFF. The example application demonstrates a method of setting the INTC_PSR registers using this array. If vector table generation is disabled in RTA-OSEK then the array is only assembled if the symbol `OS_GEN_PSC_TABLE` is defined (i.e. `-DOS_GEN_PSC_TABLE` command line option).

3.1.11 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.

Because RTA-OSEK only emits interrupt vectors for addresses 0x10000 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.

3.1.12 Addition of user ISR timing hooks

The execution time of a Category 2 ISR is not measured by RTA-OSEK in the standard build. The period that a task is interrupted not only covers the ISR execution and associated RTA-OSEK overhead but also the time to execute any higher priority tasks that have been triggered as a result of the interrupt. To make this measurement the RTA-OSEK wrappers placed around a Category 2 ISR should be replaced with user alternatives that take the measurement and perform the OS housekeeping. A user defined vector table must be used so that this user wrapper function is entered when the interrupt triggers. These user outer wrappers must also be compiled using `-Xnested-interrupts` command line option to save and restore the SRR0 and SRR1 registers.

For the Category 2 ISR `isr1` the user outer wrapper takes the form:

```
#include "osek.h"

/* function prototype for the inner wrapper */
#ifdef OS_ET_MEASURE
os_imask os_wrapper(os_t_handle);
#else
os_imask os_wrapper(void);
#endif

#ifndef OS_ET_MEASURE
/* wrapper for the standard build */
__interrupt__ void isr_entry_1(void)
{
 os_imask return_IPL;

 /* Do the start timing hook */

 /* Re-enable interrupts by setting the MSR[EE] bit
or by copying the SRR0 to the MSR to preserve the
SPE bit */

 /* Call the ISR declared as ISR(isr1) */
```

```
 osek_isr_e_isr1();

 /* Call the inner wrapper and get the old IPL
value into return_IPL */
 return_IPL = os_wrapper();

 /* Disable interrupts by clearing the MSR[EE] bit
*/

 /* Restore the previous interrupt level */
 OS_INTC_CPR = return_IPL;

 /* Do the end timing hook */
}
#endif
```

RTA-OSEK supports the measurement of execution time of Category 2 interrupts in the Timing and Extended build. If the same technique is to be used in these builds as in the standard build a user wrapper should be used in place of the default RTA-OSEK wrapper. The defined symbol OS_ET_MEASURE can be used py the C preprocessor to conditionally include code fragments as this is only present in the timing and extended builds.

```
#include "osek.h"

/* function prototype for the inner wrapper */
#ifdef OS_ET_MEASURE
os_imask os_wrapper(os_t_handle);
#else
os_imask os_wrapper(void);
#endif

#ifdef OS_ET_MEASURE
/* test wrapper for linking the TCB directly for
the timing and extended build */
__interrupt__ void isr_entry_1(void)
{
 os_imask return_IPL;

 /* Do the start timing hook */

 /* Re-enable interrupts by setting the MSR[EE] bit
or by copying the SRR0 to the MSR to preserve the
SPE bit */

 /* Call the inner wrapper and pass the TCB for
isr1 */
 return_IPL = os_wrapper(osek_interrupt_isr1);

 /* Disable interrupts by clearing the MSR[EE] bit
*/

 /* Restore the previous interrupt level */
 OS_INTC_CPR = return_IPL;
```

```
  /* Do the end timing hook */
}
#endif
```

3.2 Register Settings

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

Register	Required Value
IVPR	Interrupt vector table base address.
MSR[EE]	The EE bit should be set to enable External Interrupts.
MSR[PR]	The PR bit should not be set as RTA-OSEK expects that the processor always operates at supervisor level.
INTC_PSRn	The INTC priority select registers should contain the applicable priority for each interrupt source.
INTC_CPR	The INTC current priority register should be set to prevent Category 2 interrupts from triggering before calling StartOS() but not block any Category 1 interrupts.

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

Register	Notes
INTC_CPR	The INTC current priority register should not be manipulated after calling StartOS().
MSR[EE]	The global interrupt enable bit should not be manipulated by the user after calling StartOS().

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): 80

Timing

API max usage (bytes): 80

Extended

API max usage (bytes): 96

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

3.3.3 Stack discipline

RTA-OSEK adheres to the EABI requirements for stack discipline, in particular that the stack pointer (R1) is adjusted only once in each routine, a back-link is maintained, and the stack pointer is kept aligned to a 16-byte boundary. During start-up the user's application code should set R1 to a suitable value, in on-chip or external RAM.

Important: The initial stack pointer (R1) value must be made known in the symbol os_SP_INIT.

Typically your linker control file will need to include the line:

```
os_SP_INIT = __SP_INIT;
```

3.4 Floating point

The Freescale PowerPC e200z6 and e200z3 CPUs contain a Signal Processing Extension (SPE) auxiliary processing unit to support single-precision floating point and vector processing operations. When instructions performed on the SPE are used for more than one task or ISR, additional registers must be saved to prevent corruption of their values. The number of additional registers that must be saved depends upon the type of instructions performed in the SPE:

Single-precision floating point arithmetic: If only the Single-precision floating point instructions are used in an application then only the SPEFSCR needs to be additionally saved.

Vector processing arithmetic: If the vector processing instructions are used in an application then the CPU extends the General Purpose registers (GPRs) to 64 bits. As the top 32-bits of the GPRs are not normally saved so these must additionally be saved in addition to the SPE accumulator.

An example of how to save this floating-point context can be found in `osfptgt.c` and `osfptgt.h` in the `<RTA-OSEK location>\wr55xxvle\inc` directory. For an application to use floating-point context saving, the appropriate tasks and Category 2 interrupt service routines must be marked as using floating-point operation in the RTA-OSEK GUI. Note that the performance figures have been collected for a system not using hardware floating point.

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	**MPC5567**
Clock speed (MHz)	40
Code memory	On-chip FLASH
Read-only data memory	On-chip FLASH
Read-write data memory	On-chip RAM

4.1 Functionality

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

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

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

4.2 Hardware Resources

4.2.1 ROM and RAM Overheads

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	28	28	28	28	28	28
	ROM	146	146	146	146	146	146
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	48	48	48	48	48	48
	ROM	218	218	218	218	218	218
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	66	66	66	66	66	66
	ROM	264	264	264	264	264	264
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	44	52	n/a	44	52
ECC1, Integer task	RAM	n/a	n/a	n/a	116	116	116
	ROM	n/a	n/a	n/a	60	60	60
ECC1, floating-point task	RAM	n/a	n/a	n/a	120	120	120
	ROM	n/a	n/a	n/a	60	60	60
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	118
	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	122
	ROM	n/a	n/a	n/a	n/a	n/a	68
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	56	56	56	56	56	56
Category 2 ISR, floating-point	RAM	4	4	4	4	4	4
	ROM	84	84	84	84	84	84
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	42	42	42	42	42	42
Counter	RAM	4	4	4	4	4	4
	ROM	54	54	54	54	54	54
Message	RAM	11	11	11	51	51	51
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
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	56	64	n/a	56	64
ECC1, Integer task	RAM	n/a	n/a	n/a	128	128	128
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating-point task	RAM	n/a	n/a	n/a	132	132	132
	ROM	n/a	n/a	n/a	72	72	72
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	130
	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	134
	ROM	n/a	n/a	n/a	n/a	n/a	80

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	100	100	100	100	100	100
Category 2 ISR, floating-point	RAM	16	16	16	16	16	16
	ROM	124	124	124	124	124	124
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	42	42	42	42	42	42
Counter	RAM	4	4	4	4	4	4
	ROM	54	54	54	54	54	54
Message	RAM	11	11	11	51	51	51
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
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
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	64	72	n/a	64	72
ECC1, Integer task	RAM	n/a	n/a	n/a	132	132	132
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	136	136	136
	ROM	n/a	n/a	n/a	80	80	80
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	134
	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	138
	ROM	n/a	n/a	n/a	n/a	n/a	88
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	112	112	112	112	112	112
Category 2 ISR, floating-point	RAM	20	20	20	20	20	20
	ROM	136	136	136	136	136	136
Resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	46	46	46	46	46	46
Counter	RAM	4	4	4	4	4	4
	ROM	58	58	58	58	58	58
Message	RAM	11	11	11	51	51	51
	ROM	24	24	24	60	60	60
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
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
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.

Variant	Description
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.
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	116	176	212	120	182	240
	NS		102	158	194	106	164	222
	KL	2	56	114	154	60	120	180

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	14	14	14	14	14	14
ChainTask	SWL	1, 8	118	180	214	122	186	244
	SWH	1, 9	134	192	226	138	198	252
	NSL	8	118	180	214	122	186	244
	NSH	9	128	186	220	132	192	246
Schedule			96	96	122	96	96	122
GetTaskID			24	24	24	24	24	24
GetTaskState			82	82	82	96	96	96
EnableAllInterrupts			36	36	36	36	36	36
DisableAllInterrupts			44	44	44	44	44	44
ResumeAllInterrupts			50	50	50	50	50	50
SuspendAllInterrupts			62	62	62	62	62	62
ResumeOSInterrupts			44	44	44	44	44	44
SuspendOSInterrupts			64	64	64	64	64	64
GetResource	Task	7	20	20	24	20	20	24
	Combined	6	74	74	74	74	74	74
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	78	78	78	78	78	78
	Combined	6	158	158	158	158	158	158
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	118	118	196
	NS		n/a	n/a	n/a	94	94	172
	NS1i	10	n/a	n/a	n/a	66	n/a	n/a
	KL	2	n/a	n/a	n/a	60	60	138
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	58	58	58
GetEvent			n/a	n/a	n/a	10	10	10
WaitEvent	<default>		n/a	n/a	n/a	218	218	396
	fp	11	n/a	n/a	n/a	246	246	454
	1i	10	n/a	n/a	n/a	16	n/a	n/a
GetAlarmBase			62	62	62	62	62	62
GetAlarm			96	96	96	96	96	96
SetRelAlarm			122	122	122	122	122	122
SetAbsAlarm			136	136	136	136	136	136
CancelAlarm			92	92	92	92	92	92
InitCounter			58	58	58	58	58	58

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			76	76	76	76	76	76
osek_tick_alarm	<default>		80	80	80	80	80	80
	KL	2	36	36	36	36	36	36
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			232	232	232	232	232	232
ShutdownOS	NoHook	12	32	32	32	32	32	32
	Hook	13	48	48	48	48	48	48
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			42	42	42	42	42	42
StopCOM			20	20	20	20	20	20
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	72	72	72	172	172	172
	CCCB	15	172	172	172	172	172	172
GetMessageResource			52	52	52	52	52	52
ReleaseMessageResource			50	50	50	50	50	50
GetMessageStatus			54	54	54	54	54	54
SendMessage	SW CCCA	1, 14	90	90	90	210	210	210
	SW CCCB	1, 15	196	196	196	210	210	210
	NS CCCA	14	90	90	90	210	210	210
	NS CCCB	15	196	196	196	210	210	210
	KL CCCA	2, 14	64	64	64	180	180	180
	KL CCCB	2, 15	166	166	166	180	180	180
main_dispatch	NoHook	12	140	140	184	140	140	184
	Hook	13	178	178	220	178	178	220
sub_dispatch	B1LF	19	32	32	32	32	32	32
	B1HI	20	102	102	102	102	102	102
	B1HF	21	110	110	110	110	110	110
	B2LI	22	n/a	76	106	n/a	76	106
	B2LF	23	n/a	82	112	n/a	82	112
	B2HI	24	n/a	172	234	n/a	172	234
	B2HF	25	n/a	180	242	n/a	180	242
	E1HI	26	n/a	n/a	n/a	340	340	400
	E1HF	27	n/a	n/a	n/a	348	348	408
	E2HI	28	n/a	n/a	n/a	n/a	n/a	400

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	E2HF	29	n/a	n/a	n/a	n/a	n/a	408
ErrorHook support		16	54	54	54	54	54	54
	ServiceID	17	64	64	64	64	64	64
	Parameters	18	78	78	78	78	78	78
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	86	138	180	92	154	204
	NS		60	114	156	66	130	180
	KL	2	16	82	122	22	98	144
ChainTaskset	SWL	1, 8	50	118	158	50	124	176
	SWH	1, 9	92	146	186	92	156	204
	NSL	8	50	118	158	50	124	176
	NSH	9	86	140	180	86	150	198
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			56	56	56	56	56	56
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			56	56	56	56	56	56
TestSubTaskset			66	66	66	66	66	66
TestEquivalentTaskset			64	64	64	64	64	64
TickSchedule	SW	1	132	130	130	130	130	130
	NS		108	110	110	110	110	110
	KL	2	78	82	82	82	82	82
AdvanceSchedule	SW	1	122	120	120	120	120	120
	NS		102	100	100	100	100	100
	KL	2	74	72	72	72	72	72
StartSchedule			90	90	90	90	90	90
StopSchedule			74	74	74	74	74	74
GetScheduleStatus			100	100	100	100	100	100
GetScheduleValue			80	80	80	80	80	80
GetScheduleNext			10	10	10	10	10	10
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			6	6	6	6	6	6
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			8	8	8	8	8	8
SetArrivalpointNext			6	6	6	6	6	6

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestArrivalpointWritable			36	36	36	36	36	36
GetExecutionTime			4	4	4	4	4	4
GetLargestExecutionTime			6	6	6	6	6	6
ResetLargestExecutionTime			4	4	4	4	4	4
GetStackOffset			14	14	14	14	14	14
Stack manipulation			58	58	58	58	58	58
Interrupt support			156	156	156	156	156	156
Setjmp functions			214	214	214	214	214	214

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	116	176	212	120	182	240
	NS		102	158	194	106	164	222
	KL	2	56	114	154	60	120	180
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	14	14	14	14	14	14
ChainTask	SWL	1, 8	118	180	214	122	186	244
	SWH	1, 9	134	192	226	138	198	252
	NSL	8	118	180	214	122	186	244
	NSH	9	128	186	220	132	192	246
Schedule			118	118	144	118	118	144
GetTaskID			24	24	24	24	24	24
GetTaskState			82	82	82	96	96	96
EnableAllInterrupts			36	36	36	36	36	36
DisableAllInterrupts			44	44	44	44	44	44
ResumeAllInterrupts			50	50	50	50	50	50
SuspendAllInterrupts			62	62	62	62	62	62
ResumeOSInterrupts			44	44	44	44	44	44
SuspendOSInterrupts			64	64	64	64	64	64
GetResource	Task	7	20	20	24	20	20	24
	Combined	6	74	74	74	74	74	74

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
ReleaseResource	Task	7	100	100	100	100	100	100
	Combined	6	202	202	202	202	202	202
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	118	118	196
	NS		n/a	n/a	n/a	94	94	172
	NS1i	10	n/a	n/a	n/a	66	n/a	n/a
	KL	2	n/a	n/a	n/a	60	60	138
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	58	58	58
GetEvent			n/a	n/a	n/a	10	10	10
WaitEvent	<default>		n/a	n/a	n/a	284	284	462
	fp	11	n/a	n/a	n/a	312	312	520
	1i	10	n/a	n/a	n/a	102	n/a	n/a
GetAlarmBase			62	62	62	62	62	62
GetAlarm			96	96	96	96	96	96
SetRelAlarm			122	122	122	122	122	122
SetAbsAlarm			136	136	136	136	136	136
CancelAlarm			92	92	92	92	92	92
InitCounter			58	58	58	58	58	58
GetCounterValue			76	76	76	76	76	76
osek_tick_alarm	<default>		80	80	80	80	80	80
	KL	2	36	36	36	36	36	36
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			278	278	278	278	278	278
ShutdownOS	NoHook	12	32	32	32	32	32	32
	Hook	13	48	48	48	48	48	48
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			42	42	42	42	42	42
StopCOM			20	20	20	20	20	20
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	72	72	72	172	172	172
	CCCB	15	172	172	172	172	172	172
GetMessageResource			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	
ReleaseMessageResource			50	50	50	50	50	50
GetMessageStatus			54	54	54	54	54	54
SendMessage	SW CCCA	1, 14	90	90	90	210	210	210
	SW CCCB	1, 15	196	196	196	210	210	210
	NS CCCA	14	90	90	90	210	210	210
	NS CCCB	15	196	196	196	210	210	210
	KL CCCA	2, 14	64	64	64	180	180	180
	KL CCCB	2, 15	166	166	166	180	180	180
main_dispatch	NoHook	12	176	176	218	176	176	218
	Hook	13	212	212	254	212	212	254
sub_dispatch	B1LF	19	26	26	26	26	26	26
	B1HI	20	106	106	106	106	106	106
	B1HF	21	114	114	114	114	114	114
	B2LI	22	n/a	54	88	n/a	54	88
	B2LF	23	n/a	60	94	n/a	60	94
	B2HI	24	n/a	130	184	n/a	130	184
	B2HF	25	n/a	138	192	n/a	138	192
	E1HI	26	n/a	n/a	n/a	330	330	390
	E1HF	27	n/a	n/a	n/a	338	338	398
	E2HI	28	n/a	n/a	n/a	n/a	n/a	390
	E2HF	29	n/a	n/a	n/a	n/a	n/a	398
ErrorHook support		16	54	54	54	54	54	54
	ServiceID	17	64	64	64	64	64	64
	Parameters	18	78	78	78	78	78	78
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	84	84	84	84	84	84
Timing_termination		4	82	82	82	82	82	82
ActivateTaskset	SW	1	86	138	180	92	154	204
	NS		60	114	156	66	130	180
	KL	2	16	82	122	22	98	144
ChainTaskset	SWL	1, 8	50	118	158	50	124	176
	SWH	1, 9	92	146	186	92	156	204
	NSL	8	50	118	158	50	124	176
	NSH	9	86	140	180	86	150	198
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			56	56	56	56	56	56
AssignTaskset			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	
RemoveTaskset			56	56	56	56	56	56
TestSubTaskset			66	66	66	66	66	66
TestEquivalentTaskset			64	64	64	64	64	64
TickSchedule	SW	1	132	130	130	130	130	130
	NS		108	110	110	110	110	110
	KL	2	78	82	82	82	82	82
AdvanceSchedule	SW	1	122	120	120	120	120	120
	NS		102	100	100	100	100	100
	KL	2	74	72	72	72	72	72
StartSchedule			90	90	90	90	90	90
StopSchedule			74	74	74	74	74	74
GetScheduleStatus			100	100	100	100	100	100
GetScheduleValue			80	80	80	80	80	80
GetScheduleNext			10	10	10	10	10	10
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			6	6	6	6	6	6
GetArrivalpointTasksetRef			6	6	6	6	6	6
GetArrivalpointNext			8	8	8	8	8	8
SetArrivalpointNext			6	6	6	6	6	6
TestArrivalpointWritable			36	36	36	36	36	36
GetExecutionTime			116	116	116	116	116	116
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			10	10	10	10	10	10
GetStackOffset			14	14	14	14	14	14
Stack manipulation			58	58	58	58	58	58
Interrupt support			152	152	152	152	152	152
Setjmp functions			214	214	214	214	214	214

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	242	296	336	244	302	362
	NS		300	354	394	302	360	422
	KL	2	172	224	258	174	230	282
TerminateTask	LExt	3	118	118	118	118	118	118
	H	5	138	138	138	138	138	138
ChainTask	SWL	1, 8	282	346	382	284	352	410
	SWH	1, 9	306	362	398	308	368	424
	NSL	8	348	412	450	350	420	478
	NSH	9	366	422	458	368	428	486
Schedule			232	232	258	232	232	258
GetTaskID			40	40	40	40	40	40
GetTaskState			232	232	232	234	234	234
EnableAllInterrupts			62	62	62	62	62	62
DisableAllInterrupts			60	60	60	60	60	60
ResumeAllInterrupts			104	104	104	104	104	104
SuspendAllInterrupts			78	78	78	78	78	78
ResumeOSInterrupts			98	98	98	98	98	98
SuspendOSInterrupts			90	90	90	90	90	90
GetResource	Task	7	336	336	294	336	336	294
	Combined	6	320	320	320	320	320	320
	CLEx	3	280	280	280	280	280	280
ReleaseResource	Task	7	316	316	316	316	316	316
	Combined	6	412	412	412	412	412	412
	CLEx	3	278	278	278	278	278	278
SetEvent	SW	1	n/a	n/a	n/a	280	280	360
	NS		n/a	n/a	n/a	336	336	418
	NS1i	10	n/a	n/a	n/a	228	n/a	n/a
	KL	2	n/a	n/a	n/a	204	204	278
	KL1i	2, 10	n/a	n/a	n/a	166	n/a	n/a
ClearEvent			n/a	n/a	n/a	136	136	136
GetEvent			n/a	n/a	n/a	158	158	158
WaitEvent	<default>		n/a	n/a	n/a	370	370	526
	fp	11	n/a	n/a	n/a	398	398	584

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	198	n/a	n/a
GetAlarmBase			172	172	172	172	172	172
GetAlarm			170	170	170	170	170	170
SetRelAlarm			220	220	220	220	220	220
SetAbsAlarm			246	246	246	246	246	246
CancelAlarm			162	162	162	162	162	162
InitCounter			212	212	212	212	212	212
GetCounterValue			184	184	184	184	184	184
osek_tick_alarm	<default>		122	122	122	122	122	122
	KL	2	36	36	36	36	36	36
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			292	292	292	292	292	292
ShutdownOS	NoHook	12	42	42	42	42	42	42
	Hook	13	58	58	58	58	58	58
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			62	62	62	62	62	62
StopCOM			46	46	46	46	46	46
ReadFlag			34	34	34	34	34	34
ResetFlag			36	36	36	36	36	36
ReceiveMessage	CCCA	14	164	164	164	260	260	260
	CCCB	15	260	260	260	260	260	260
GetMessageResource			102	102	102	102	102	102
ReleaseMessageResource			106	106	106	106	106	106
GetMessageStatus			116	116	116	116	116	116
SendMessage	SW CCCA	1, 14	198	198	198	310	310	310
	SW CCCB	1, 15	296	296	296	310	310	310
	NS CCCA	14	198	198	198	310	310	310
	NS CCCB	15	296	296	296	310	310	310
	KL CCCA	2, 14	146	146	146	256	256	256
	KL CCCB	2, 15	242	242	242	256	256	256
main_dispatch	NoHook	12	176	176	218	176	176	218
	Hook	13	212	212	254	212	212	254
sub_dispatch	B1LF	19	26	26	26	26	26	26
	B1HI	20	106	106	106	106	106	106
	B1HF	21	114	114	114	114	114	114

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	B2LI	22	n/a	54	88	n/a	54	88
	B2LF	23	n/a	60	94	n/a	60	94
	B2HI	24	n/a	130	184	n/a	130	184
	B2HF	25	n/a	138	192	n/a	138	192
	E1HI	26	n/a	n/a	n/a	330	330	390
	E1HF	27	n/a	n/a	n/a	338	338	398
	E2HI	28	n/a	n/a	n/a	n/a	n/a	390
	E2HF	29	n/a	n/a	n/a	n/a	n/a	398
ErrorHook support		16	114	114	114	114	114	114
	ServiceID	17	124	124	124	124	124	124
	Parameters	18	144	144	144	144	144	144
validity_checks		3	40	40	40	40	40	40
Timing_dispatch		4	84	84	84	84	84	84
Timing_termination		4	82	82	82	82	82	82
ActivateTaskset	SW	1	322	372	414	336	398	460
	NS		376	426	468	390	450	520
	KL	2	246	282	324	256	306	364
ChainTaskset	SWL	1, 8	374	430	468	378	444	512
	SWH	1, 9	416	474	518	420	490	554
	NSL	8	446	508	548	450	522	584
	NSH	9	486	538	580	490	556	618
GetTasksetRef			134	134	134	134	134	134
MergeTaskset			244	244	244	244	244	244
AssignTaskset			158	158	158	158	158	158
RemoveTaskset			244	244	244	244	244	244
TestSubTaskset			274	274	274	274	274	274
TestEquivalentTaskset			272	272	272	272	272	272
TickSchedule	SW	1	294	238	238	238	238	238
	NS		348	306	306	306	306	306
	KL	2	232	174	174	174	174	174
AdvanceSchedule	SW	1	302	246	246	246	246	246
	NS		358	314	314	314	314	314
	KL	2	240	180	180	180	180	180
StartSchedule			228	228	228	228	228	228
StopSchedule			180	180	180	180	180	180
GetScheduleStatus			210	210	210	210	210	210
GetScheduleValue			172	172	172	172	172	172

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleNext			86	86	86	86	86	86
SetScheduleNext			166	166	166	166	166	166
GetArrivalpointDelay			126	126	126	126	126	126
SetArrivalpointDelay			140	140	140	140	140	140
GetArrivalpointTasksetRef			124	124	124	124	124	124
GetArrivalpointNext			126	126	126	126	126	126
SetArrivalpointNext			172	172	172	172	172	172
TestArrivalpointWritable			146	146	146	146	146	146
GetExecutionTime			166	166	166	166	166	166
GetLargestExecutionTime			106	106	106	106	106	106
ResetLargestExecutionTime			102	102	102	102	102	102
GetStackOffset			14	14	14	14	14	14
Stack manipulation			58	58	58	58	58	58
Interrupt support			152	152	152	152	152	152
Setjmp functions			214	214	214	214	214	214

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.
16	Linked only if `USEGETSERVICEID = FALSE`

Number	Note
	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

4.2.4 Reserved Hardware Resources

4.3 Performance

4.3.1 Execution Times for RTA-OSEK API Calls

The following tables give the execution time (in CPU cycles) for each API call. (Note that: (1) the OSEK COM class was set to CCCA for systems without events and to CCCB for systems with events; (2) `ShutdownOS()` enters an infinite loop; the execution time for `ShutdownOS()` reported below is the time up to the point at which `ShutdownOS()` calls `ShutdownHook()`).

Standard

Configuration		Application Uses					
	Events	No			Yes		
	Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	122	178	222	121	157	218
	NS	111	165	217	108	145	212
	KL	63	130	167	62	101	166
TerminateTask	LExt	0	0	0	0	0	0
	H	253	254	264	256	266	266
ChainTask	SWL	389	464	565	453	500	612
	SWH	503	559	653	561	599	703
	NSL	392	461	565	453	502	617
	NSH	500	545	652	567	596	699
Schedule	SW	135	138	156	135	147	155
GetTaskID		41	39	41	40	35	35
GetTaskState		120	118	114	123	131	126
EnableAllInterrupts		51	51	52	51	54	51
DisableAllInterrupts		57	55	56	57	53	54
ResumeAllInterrupts		63	61	62	63	64	62
SuspendAllInterrupts		68	67	68	68	69	68
ResumeOSInterrupts		63	61	62	63	64	62
SuspendOSInterrupts		68	67	68	68	69	68
GetResource	Task	54	57	64	54	56	58
	Combined	98	98	95	95	102	99
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	111	106	113	111	111	109
	Combined	173	172	173	174	176	175
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	137	143	157
	NS	n/a	n/a	n/a	130	126	141
	KL	n/a	n/a	n/a	90	89	88
ClearEvent		n/a	n/a	n/a	88	89	89
GetEvent		n/a	n/a	n/a	38	39	40
WaitEvent	<default>	n/a	n/a	n/a	707	705	761
	fp	n/a	n/a	n/a	729	716	775
GetAlarmBase		118	115	113	116	120	120
GetAlarm		130	131	134	134	138	138

Configuration		Application Uses					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
SetRelAlarm		141	146	145	149	145	145
SetAbsAlarm		160	166	165	168	164	164
CancelAlarm		109	106	108	107	109	109
InitCounter		88	87	91	94	86	85
GetCounterValue		111	117	112	112	114	114
osek_tick_alarm	<default>	107	106	105	105	105	105
	KL	48	46	45	43	46	46
osek_incr_counter		22	22	22	24	24	24
GetActiveApplicationMode		11	11	11	12	12	12
StartOS		1212	1211	1225	1291	1213	1188
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	68	68	66	68	70	73
InitCOM		22	20	20	19	18	19
CloseCOM		18	18	20	19	19	20
StartCOM		72	71	68	112	116	109
StopCOM		33	30	32	32	35	37
ReadFlag		n/a	n/a	n/a	23	21	21
ResetFlag		n/a	n/a	n/a	21	21	21
ReceiveMessage		98	105	95	343	364	366
GetMessageResource		n/a	n/a	n/a	147	149	147
ReleaseMessageResource		n/a	n/a	n/a	196	191	196
GetMessageStatus		n/a	n/a	n/a	71	75	66
SendMessage	SW	251	309	352	499	544	603
	NS	233	289	338	493	550	616
	KL	153	214	258	402	448	512
ActivateTaskset	SW	103	717	808	106	834	887
	NS	85	662	784	84	623	799
	KL	33	593	709	38	808	878
	SW2	103	717	808	106	834	887
	NS2	85	662	784	84	623	799
	KL2	33	593	709	38	808	878
ChainTaskset	SWL	359	969	1054	426	975	1187
	SWH	509	1073	1253	570	1084	1394
	NSL	373	946	1117	417	980	1261
	NSH	502	1069	1184	567	1275	1349
GetTasksetRef		32	32	33	32	32	32

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		87	93	96	87	93	87
AssignTaskset		30	30	30	30	29	29
RemoveTaskset		84	90	92	84	89	83
TestSubTaskset		93	99	96	93	98	95
TestEquivalentTaskset		93	90	92	93	89	94
TickSchedule	SW	184	767	899	228	1002	1065
	NS	160	757	871	197	991	1060
	KL	122	720	831	161	952	1019
	SW2	183	766	899	228	985	1054
	NS2	159	757	872	197	975	1049
	KL2	122	720	832	161	936	1008
AdvanceSchedule	SW	160	740	854	183	979	1045
	NS	142	726	843	173	963	1029
	KL	102	688	802	130	920	986
	SW2	160	740	854	182	963	1034
	NS2	141	726	843	172	947	1018
	KL2	102	688	802	130	904	975
StartSchedule		152	141	150	152	136	135
StopSchedule		126	126	129	126	128	125
GetScheduleStatus		137	137	138	137	139	139
GetScheduleValue		127	135	126	127	124	123
GetScheduleNext		38	39	37	38	37	37
SetScheduleNext		34	34	34	34	34	35
GetArrivalpointDelay		36	35	33	35	33	33
SetArrivalpointDelay		26	29	27	26	30	27
GetArrivalpointTasksetRef		24	27	23	24	26	26
GetArrivalpointNext		29	29	30	29	31	31
SetArrivalpointNext		27	30	26	27	26	29
TestArrivalpointWritable		40	35	35	35	35	43
GetExecutionTime		19	19	21	20	21	21
GetLargestExecutionTime		31	31	30	31	30	30
ResetLargestExecutionTime		24	25	24	24	25	24
GetStackOffset		27	27	26	27	26	26

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	124	177	221	132	159	228
	NS	107	166	214	114	150	204
	KL	65	129	169	70	102	172
TerminateTask	LExt	0	0	0	0	0	0
	H	505	502	499	510	498	496
ChainTask	SWL	680	756	858	744	810	917
	SWH	776	847	938	852	895	993
	NSL	683	760	853	753	809	919
	NSH	776	839	939	857	891	991
Schedule	SW	139	139	157	139	146	153
GetTaskID		37	37	32	37	36	37
GetTaskState		111	117	114	131	126	130
EnableAllInterrupts		52	52	53	54	50	55
DisableAllInterrupts		56	56	52	53	54	54
ResumeAllInterrupts		62	62	63	64	63	65
SuspendAllInterrupts		68	68	68	69	68	70
ResumeOSInterrupts		62	62	63	64	63	65
SuspendOSInterrupts		68	68	68	69	68	70
GetResource	Task	60	60	60	56	58	61
	Combined	95	95	99	102	95	98
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	114	114	113	110	113	111
	Combined	168	168	171	172	171	175
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	142	135	144
	NS	n/a	n/a	n/a	120	129	131
	KL	n/a	n/a	n/a	92	92	97
ClearEvent		n/a	n/a	n/a	86	90	89
GetEvent		n/a	n/a	n/a	39	39	39
WaitEvent	<default>	n/a	n/a	n/a	943	914	990
	fp	n/a	n/a	n/a	952	929	1006
GetAlarmBase		112	115	113	119	113	116
GetAlarm		130	131	133	134	134	131

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		142	143	149	144	148	147
SetAbsAlarm		166	165	164	160	166	168
CancelAlarm		103	105	106	107	109	105
InitCounter		88	88	88	88	90	87
GetCounterValue		118	118	117	117	111	118
osek_tick_alarm	<default>	102	105	101	105	106	107
	KL	46	45	45	47	46	43
osek_incr_counter		22	22	24	24	22	25
GetActiveApplicationMode		11	11	12	12	11	12
StartOS		2864	3084	3186	3182	2908	2908
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	67	70	71	73	68	71
InitCOM		21	20	18	18	23	19
CloseCOM		20	20	22	18	21	20
StartCOM		74	71	72	115	110	116
StopCOM		34	32	31	31	30	32
ReadFlag		n/a	n/a	n/a	21	22	21
ResetFlag		n/a	n/a	n/a	21	19	21
ReceiveMessage		97	95	105	347	360	364
GetMessageResource		n/a	n/a	n/a	146	150	146
ReleaseMessageResource		n/a	n/a	n/a	191	193	196
GetMessageStatus		n/a	n/a	n/a	73	69	70
SendMessage	SW	245	309	353	517	542	620
	NS	236	289	340	500	540	592
	KL	158	214	255	397	453	518
ActivateTaskset	SW	104	719	836	111	633	849
	NS	83	663	725	93	616	796
	KL	35	592	643	36	581	764
	SW2	104	719	836	111	633	849
	NS2	83	663	725	93	616	796
	KL2	35	592	643	36	581	764
ChainTaskset	SWL	649	1258	1414	716	1454	1481
	SWH	787	1359	1479	857	1447	1639
	NSL	648	1238	1406	730	1318	1510
	NSH	782	1354	1475	858	1381	1617
GetTasksetRef		32	32	32	32	32	30

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		93	93	95	95	89	94
AssignTaskset		30	29	28	30	27	29
RemoveTaskset		89	89	92	91	86	92
TestSubTaskset		96	98	93	99	89	96
TestEquivalentTaskset		89	89	90	92	93	91
TickSchedule	SW	187	767	829	212	786	951
	NS	161	763	806	202	765	928
	KL	119	720	776	163	737	900
	SW2	187	767	829	212	765	946
	NS2	161	763	806	202	744	924
	KL2	120	720	776	163	716	896
AdvanceSchedule	SW	159	740	788	197	748	914
	NS	140	725	778	174	738	900
	KL	102	688	742	133	702	865
	SW2	160	741	789	197	727	909
	NS2	140	726	779	174	717	895
	KL2	102	688	742	133	681	860
StartSchedule		139	139	146	136	146	144
StopSchedule		128	128	124	128	124	123
GetScheduleStatus		138	138	141	139	141	141
GetScheduleValue		134	134	133	124	133	133
GetScheduleNext		38	38	38	37	38	37
SetScheduleNext		34	34	33	34	33	34
GetArrivalpointDelay		33	33	35	34	35	33
SetArrivalpointDelay		30	27	24	30	24	25
GetArrivalpointTasksetRef		26	26	25	23	25	25
GetArrivalpointNext		31	31	29	31	29	29
SetArrivalpointNext		26	29	24	29	24	26
TestArrivalpointWritable		35	43	42	35	42	41
GetExecutionTime		148	143	144	146	141	144
GetLargestExecutionTime		50	50	49	49	50	49
ResetLargestExecutionTime		41	40	41	40	41	39
GetStackOffset		27	26	23	26	23	28

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	419	483	537	423	456	501
	NS	480	547	584	489	519	568
	KL	352	423	453	362	397	444
TerminateTask	LExt	580	585	575	588	582	581
	H	670	672	673	671	669	672
ChainTask	SWL	1081	1148	1255	1151	1208	1306
	SWH	1196	1261	1345	1268	1306	1400
	NSL	1147	1229	1317	1234	1275	1371
	NSH	1257	1314	1407	1322	1355	1453
Schedule	SW	222	222	232	231	230	232
GetTaskID		51	51	46	48	47	49
GetTaskState		425	444	432	436	440	430
EnableAllInterrupts		67	67	64	68	67	69
DisableAllInterrupts		65	65	64	63	62	68
ResumeAllInterrupts		90	90	89	91	90	95
SuspendAllInterrupts		77	77	69	80	79	72
ResumeOSInterrupts		90	90	89	91	90	95
SuspendOSInterrupts		77	77	69	80	79	72
GetResource	Task	792	802	408	857	858	457
	Combined	383	393	377	440	438	430
	CLEx	409	420	415	478	475	470
ReleaseResource	Task	402	412	402	450	450	465
	Combined	397	407	405	466	465	463
	CLEx	392	405	377	459	459	441
SetEvent	SW	n/a	n/a	n/a	443	467	458
	NS	n/a	n/a	n/a	480	487	495
	KL	n/a	n/a	n/a	408	392	408
ClearEvent		n/a	n/a	n/a	155	156	150
GetEvent		n/a	n/a	n/a	357	358	354
WaitEvent	<default>	n/a	n/a	n/a	1070	1072	1101
	fp	n/a	n/a	n/a	1080	1081	1116
GetAlarmBase		325	340	328	333	325	321
GetAlarm		337	351	343	336	340	343

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		382	396	372	378	386	363
SetAbsAlarm		400	414	394	391	403	383
CancelAlarm		312	326	317	307	316	316
InitCounter		567	579	572	568	568	567
GetCounterValue		298	310	304	300	299	300
osek_tick_alarm	<default>	153	153	149	149	150	147
	KL	45	45	47	46	50	49
osek_incr_counter		25	25	23	23	22	22
GetActiveApplicationMode		12	12	11	11	13	13
StartOS		3124	3112	3018	3004	3198	3215
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	74	75	74	75	75	71
InitCOM		23	25	18	20	23	22
CloseCOM		22	21	19	22	18	20
StartCOM		94	94	96	135	134	140
StopCOM		42	48	45	50	49	49
ReadFlag		n/a	n/a	n/a	43	40	41
ResetFlag		n/a	n/a	n/a	44	43	44
ReceiveMessage		257	275	261	520	510	516
GetMessageResource		n/a	n/a	n/a	658	657	655
ReleaseMessageResource		n/a	n/a	n/a	656	656	656
GetMessageStatus		n/a	n/a	n/a	215	212	199
SendMessage	SW	701	775	823	965	997	1050
	NS	767	844	868	1025	1050	1104
	KL	581	662	674	846	881	912
ActivateTaskset	SW	734	1316	1435	778	1377	1441
	NS	815	1419	1501	796	1529	1554
	KL	698	1219	1302	674	1233	1466
	SW2	734	1316	1435	778	1377	1441
	NS2	815	1419	1501	796	1529	1554
	KL2	698	1219	1302	674	1233	1466
ChainTaskset	SWL	1456	2158	2162	1520	2084	2249
	SWH	1562	2137	2274	1627	2189	2380
	NSL	1505	2074	2324	1552	2061	2326
	NSH	1596	2356	2357	1655	2270	2443
GetTasksetRef		319	335	326	325	320	319

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		204	199	206	194	188	192
AssignTaskset		122	115	120	113	108	113
RemoveTaskset		181	184	180	194	202	197
TestSubTaskset		205	206	204	200	209	200
TestEquivalentTaskset		200	201	202	200	203	201
TickSchedule	SW	300	1479	1560	927	1521	1745
	NS	355	1537	1613	982	1575	1802
	KL	230	1407	1486	851	1447	1674
	SW2	300	1479	1560	927	1484	1718
	NS2	355	1537	1613	982	1538	1775
	KL2	230	1407	1486	851	1410	1647
AdvanceSchedule	SW	288	1454	1540	906	1495	1716
	NS	347	1512	1592	957	1557	1781
	KL	218	1378	1457	834	1433	1647
	SW2	288	1454	1540	906	1458	1689
	NS2	347	1512	1592	957	1520	1754
	KL2	218	1378	1457	834	1396	1620
StartSchedule		250	250	238	248	250	247
StopSchedule		181	181	181	182	180	183
GetScheduleStatus		194	194	201	199	197	198
GetScheduleValue		184	184	189	191	190	196
GetScheduleNext		73	73	72	70	70	71
SetScheduleNext		137	137	133	136	135	143
GetArrivalpointDelay		103	101	95	97	97	100
SetArrivalpointDelay		124	129	118	114	113	123
GetArrivalpointTasksetRef		81	79	81	78	80	82
GetArrivalpointNext		83	86	87	88	87	83
SetArrivalpointNext		141	139	142	145	146	142
TestArrivalpointWritable		97	93	93	90	89	97
GetExecutionTime		192	190	194	195	194	195
GetLargestExecutionTime		304	318	309	305	305	301
ResetLargestExecutionTime		290	304	296	292	292	289
GetStackOffset		24	24	25	26	28	28

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	74	74	74	74	74	74
	Cat 2	107	108	106	111	106	105

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	74	74	74	74	74	74
	Cat 2	284	287	295	287	295	290

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	74	74	74	74	74	74
	Cat 2	283	283	293	289	291	289

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.

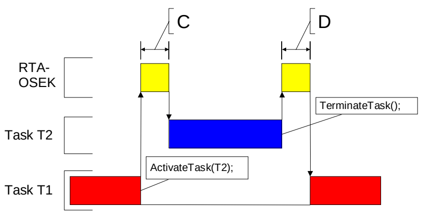

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

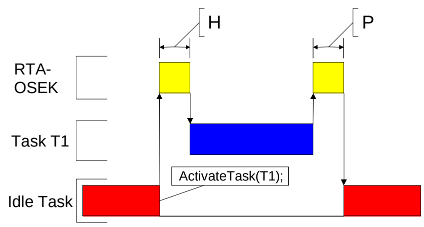

Figure 3: Task Activation from Idle Task

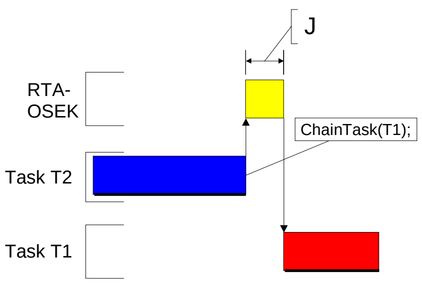

Figure 2: Task Chaining

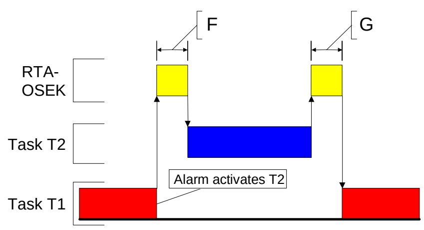

Figure 4: Task Activation from an Alarm

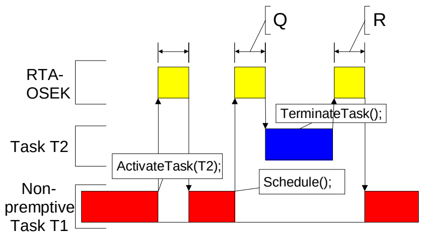

Figure 5: Non-Premptive Task Calls Schedule()

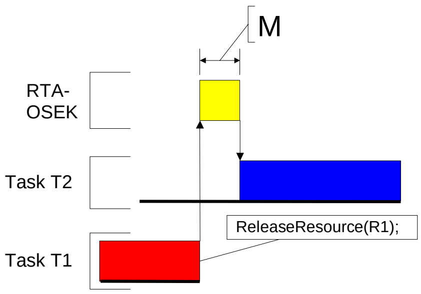

Figure 6: Blocked Task Activated by ReleaseResource()

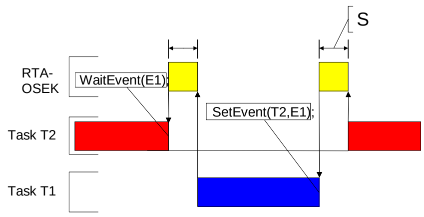

Figure 7: Waiting Task Activated by SetEvent()

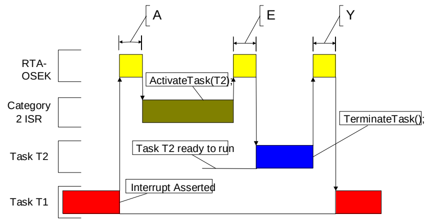

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	127	192	223	129	200	233
Figure 1: D	Heavy, Basic/Extended	255	309	344	319	327	352
ChainTask	Light, Basic	260	364	463	266	368	480
Figure 2: J	Heavy, Basic/Extended	654	793	920	718	821	949
Pre-emption	Light, Basic	196	275	379	197	275	399
Figure 1: C	Heavy, Basic/Extended	338	388	493	395	418	548
From idle task	Light, Basic	197	276	380	198	276	400
Figure 3: H	Heavy, Basic/Extended	339	389	494	396	419	549
Triggered by alarm	Light, Basic	369	451	550	367	446	569
Figure 4: F	Heavy, Basic/Extended	508	561	667	565	587	720
Schedule	Light, Basic	188	211	291	190	215	296
Figure 5: Q	Heavy, Basic/Extended	330	324	405	388	381	464
Release resource	Light, Basic	201	221	292	203	223	291
Figure 6: M	Heavy, Basic/Extended	343	334	406	401	389	459
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	738	726	895
From category 2 ISR	Light, Basic	196	215	290	200	220	286
Figure 8: E	Heavy, Basic/Extended	338	328	404	398	386	454

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	382	415	449	381	411	453
Figure 1: D	Heavy, Basic/Extended	504	528	559	554	540	571
ChainTask	Light, Basic	567	652	753	559	655	765
Figure 2: J	Heavy, Basic/Extended	1188	1288	1414	1247	1308	1444
Pre-emption	Light, Basic	350	416	522	351	422	545
Figure 1: C	Heavy, Basic/Extended	475	532	640	549	585	706
From idle task	Light, Basic	351	417	524	352	424	545
Figure 3: H	Heavy, Basic/Extended	476	533	642	550	587	706
Triggered by alarm	Light, Basic	521	592	689	522	596	721
Figure 4: F	Heavy, Basic/Extended	643	705	810	718	761	878
Schedule	Light, Basic	343	354	433	342	356	434
Figure 5: Q	Heavy, Basic/Extended	468	470	552	540	540	615
Release resource	Light, Basic	351	362	435	351	369	433
Figure 6: M	Heavy, Basic/Extended	476	478	554	549	553	614
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	845	815	989
From category 2 ISR	Light, Basic	608	612	689	605	615	681
Figure 8: E	Heavy, Basic/Extended	729	720	807	795	798	861

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	573	622	646	578	617	651
Figure 1: D	Heavy, Basic/Extended	674	703	727	714	716	737
ChainTask	Light, Basic	956	1043	1158	958	1043	1160
Figure 2: J	Heavy, Basic/Extended	1770	1881	2007	1812	1884	2014
Pre-emption	Light, Basic	617	695	818	621	686	803
Figure 1: C	Heavy, Basic/Extended	753	811	926	825	862	956
From idle task	Light, Basic	619	697	818	621	687	804
Figure 3: H	Heavy, Basic/Extended	755	813	926	825	863	957
Triggered by alarm	Light, Basic	837	915	1037	837	901	1021
Figure 4: F	Heavy, Basic/Extended	973	1031	1141	1037	1079	1172
Schedule	Light, Basic	402	415	495	403	417	495
Figure 5: Q	Heavy, Basic/Extended	538	531	603	607	609	671
Release resource	Light, Basic	588	611	668	652	666	730
Figure 6: M	Heavy, Basic/Extended	724	727	776	856	858	906
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1120	1153	1280
From category 2 ISR	Light, Basic	660	670	732	655	660	734
Figure 8: E	Heavy, Basic/Extended	795	782	839	855	848	909

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	128	128	144	128	128	144
BCC1 lightweight, floating-point	144	144	160	144	144	160
BCC1 heavyweight, integer	240	240	256	240	240	256
BCC1 heavyweight, floating-point	240	240	256	240	240	256
BCC2 lightweight, integer	n/a	144	160	n/a	144	160
BCC2 lightweight, floating-point	n/a	144	160	n/a	144	160
BCC2 heavyweight, integer	n/a	240	272	n/a	240	272
BCC2 heavyweight, floating-point	n/a	240	272	n/a	240	272
ECC1 heavyweight, integer	n/a	n/a	n/a	288	288	304
ECC1 heavyweight, floating-point	n/a	n/a	n/a	288	288	304
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	304
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	304
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	144	144	144	144	144	144
BCC1 lightweight, floating-point	160	160	160	160	160	160
BCC1 heavyweight, integer	256	256	256	256	256	256
BCC1 heavyweight, floating-point	256	256	256	256	256	256
BCC2 lightweight, integer	n/a	160	160	n/a	160	160
BCC2 lightweight, floating-point	n/a	160	160	n/a	160	160
BCC2 heavyweight, integer	n/a	256	272	n/a	256	272
BCC2 heavyweight, floating-point	n/a	256	272	n/a	256	272
ECC1 heavyweight, integer	n/a	n/a	n/a	304	304	304
ECC1 heavyweight, floating-point	n/a	n/a	n/a	304	304	304
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	304
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	304

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		176	176	176	176	176	176
BCC1 lightweight, floating-point		192	192	192	192	192	192
BCC1 heavyweight, integer		304	304	304	304	304	304
BCC1 heavyweight, floating-point		304	304	304	304	304	304
BCC2 lightweight, integer		n/a	192	192	n/a	192	192
BCC2 lightweight, floating-point		n/a	192	192	n/a	192	192
BCC2 heavyweight, integer		n/a	304	304	n/a	304	304
BCC2 heavyweight, floating-point		n/a	304	304	n/a	304	304
ECC1 heavyweight, integer		n/a	n/a	n/a	336	336	336
ECC1 heavyweight, floating-point		n/a	n/a	n/a	336	336	336
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	336
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	336
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		176	176	176	176	176	176
BCC1 lightweight, floating-point		192	192	192	192	192	192
BCC1 heavyweight, integer		304	304	304	304	304	304
BCC1 heavyweight, floating-point		304	304	304	304	304	304
BCC2 lightweight, integer		n/a	192	192	n/a	192	192
BCC2 lightweight, floating-point		n/a	192	192	n/a	192	192
BCC2 heavyweight, integer		n/a	304	304	n/a	304	304
BCC2 heavyweight, floating-point		n/a	304	304	n/a	304	304
ECC1 heavyweight, integer		n/a	n/a	n/a	336	336	336
ECC1 heavyweight, floating-point		n/a	n/a	n/a	336	336	336
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	336
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	336

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		176	176	176	176	176	176
BCC1 lightweight, floating-point		192	192	192	192	192	192
BCC1 heavyweight, integer		304	304	304	304	304	304
BCC1 heavyweight, floating-point		304	304	304	304	304	304
BCC2 lightweight, integer		n/a	192	192	n/a	192	192
BCC2 lightweight, floating-point		n/a	192	192	n/a	192	192
BCC2 heavyweight, integer		n/a	304	304	n/a	304	304
BCC2 heavyweight, floating-point		n/a	304	304	n/a	304	304
ECC1 heavyweight, integer		n/a	n/a	n/a	336	336	336
ECC1 heavyweight, floating-point		n/a	n/a	n/a	336	336	336
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	336
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	336
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		176	176	176	176	176	176
BCC1 lightweight, floating-point		192	192	192	192	192	192
BCC1 heavyweight, integer		304	304	304	304	304	304
BCC1 heavyweight, floating-point		304	304	304	304	304	304
BCC2 lightweight, integer		n/a	192	192	n/a	192	192
BCC2 lightweight, floating-point		n/a	192	192	n/a	192	192
BCC2 heavyweight, integer		n/a	304	304	n/a	304	304
BCC2 heavyweight, floating-point		n/a	304	304	n/a	304	304
ECC1 heavyweight, integer		n/a	n/a	n/a	336	336	336
ECC1 heavyweight, floating-point		n/a	n/a	n/a	336	336	336
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	336
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	336

Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found on the ETAS Group website www.etasgroup.com.