RTA-OSEK

Binding Manual: MPC55xxVLE/WindRiver

Contact Details

ETAS Group www.etasgroup.com	**ETAS GmbH** 70469 Stuttgart, Germany Tel.:+49 711 89661-0 Fax:+49 711 89661-300 sales.de@etas.com
ETAS Inc. Ann Arbor, MI 48103, USA Tel.: +1 888 ETAS INC Fax: +1 734 997 9449 sales.us@etas.com	**ETAS S.A.S.** 94588 Rungis Cedex, France Tel.: +33 (1) 56 70 00 50 Fax: +33 (1) 56 70 00 51 sales.fr@etas.com
ETAS K.K. Yokohama 220-6217, Japan Tel.: +81 45 222-0900 Fax: +81 45 222-0956 sales.jp@etas.com	**ETAS Ltd.** Derby DE21 4SU, UK Tel.: +44 1332 253770 Fax: +44 1332 253779 sales.uk@etas.com
ETAS Korea Co. Ltd. Seoul 137-889, Korea Tel.: +82 2 5747-016 Fax: +82 2 5747-120 sales.kr@etas.com	**ETAS (Shanghai) Co., Ltd.** Shanghai 200120, P.R. China Tel.: +86 21 5037 2220 Fax: +86 21 5037 2221 sales.cn@etas.com
ETAS in Italy 10135 TORINO, Italy Tel.: +39 011 3285 988 Fax: +39 (011) 3285 256 sales.it@etas.com	**ETAS Automotive India Pvt. Ltd.** Bangalore 560 068, India Tel.: +91 80 4191 2585 Fax: +91 80 4191 2586 sales.in@etas.com
ETAS in Brazil CEP-05802-140 São Paulo, Brazil Tel.: +55 11 2162-0252 sales.br@etas.com	**ETAS in Russia** Moscow, 129515, Russia Tel.: +7 495 937 0400 998 sales.ru@etas.com

Copyright Notice

Version: RM00089-003

Disclaimer

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

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

Trademarks

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

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Continental Automotive GmbH.

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

Contents

1 About this Guide

This guide provides target-specific information for the MPC55xxVLE/WindRiver port of ETAS' RTA-OSEK. It supplements the more general information in the *RTA-OSEK User Guide*.

A port is defined as a specific target microcontroller/target toolchain pairing. This guide tells you about integration issues with your target toolchain and issues that you need to be aware of when using RTA-OSEK on your target hardware. Port specific parameters of implementation are also provided, giving the RAM and ROM requirements for each object in the RTA-OSEK Component and execution times for each API call to the RTA-OSEK Component.

1.1 Who Should Read this Guide?

The reader should have an understanding of real time embedded programming in an OSEK context. You should read this guide if you want to know low-level technical information to integrate the RTA-OSEK Component into your application.

1.2 Conventions

Important: Notes that appear like this contain important information that you need to be aware of. Make sure that you read them carefully and that you follow any instructions that you are given.

Portability: Notes that appear like this describe things that you will need to know if you want to write code that will work on any processor running the RTA-OSEK Component.

Program code, file names, C types and symbols, and RTA-OSEK API call names all appear in the `courier` typeface. When the name of an object is made available to the programmer the name also appears in the `courier` typeface, so, for example, a task named Task1 appears as a task handle called `Task1`.

2 Toolchain Issues

This chapter contains important details about RTA-OSEK and your toolchain. A port of the RTA-OSEK Component is specific to both the target hardware and a specific version of the compiler toolchain. You must make sure that you build your application with the supported toolchain.

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

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 River (Diab) Compiler for PowerPC
Version	5.8.0.0

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 River (Diab) Assembler for the PowerPC
Version	5.8.0.0

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. Initialized during StartOS(). Can be located in 'far' RAM.
os_pir2	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. Initialized during StartOS(). Must be placed in the compiler SDA (near addressing)
os_pnir2	RAM	RTA-OSEK near initialized data. Must be initialized 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; some entries are zeroed during StartOS().
os_cntr	RAM	RTA-OSEK near initialized data for interrupt control. Must be zeroed during C-startup. Must be placed in the compiler SDA (near addressing)
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 for MPC55xxVLE/WindRiver. 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
0	0	1
1-15	1-15	1
16	15	0

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 Offset	Legality
0x0 to 0x22	The CPU vectors only handle Category 1 ISRs
0x10000 to the maximum INTC vector supported in 0x10 steps or 0x4 steps for z0/z1 core variants	The INTC vectors can handle Category 1 and 2 ISRs

The valid base addresses for the vector table are:

Register	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 Offset	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)

                        ; Form INTC_IACKR address
 lis   r3,%hi(0xFFF48010)
 ori   r3,r3,%lo(0xFFF48010)
 lwz   r3,0x0(r3)       ; Load INTC_IACKR, which
                        ; clears request to CPU
 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
 stw   r0,12(r1)         ; allow nested interrupts
 mfspr r0,srr1
 stw   r0,16(r1)

                         ; Increment the interrupt
                         ; counter before enabling
                         ; global interrupts
 stw   r4,44(r1)
 lwz   r4,os_isr_count@sdarx(r0)
 addi  r4,r4,1
 stw   r4,os_isr_count@sdarx(r0)

 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   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,r0,%hiadj(0xFFF48008)
stw   r3,%lo(0xFFF48008)(r4)

                      ; Decrement the interrupt
                      ; counter after enabling
                      ; global interrupts
lwz   r4,os_isr_count@sdarx(r0)
addi  r4,r4,-1
stw   r4,os_isr_count@sdarx(r0)
```

```
  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.

For traditional variants the IVOR4 register should hold the lower 2-bytes of the address of the SW vector mode common interrupt exception handler. For z0/z1 core variants the IVOR4 register has been replaced by an interrupt vector location as with all other CPU interrupts. These operate in the same way as for the INTC interrupt vectors and should therefore be a branch instruction to the SW vector mode common interrupt exception handler. See the next section for an example of a typical IVOR interrupt table.

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 (Traditional Variants)

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.

CPU interrupts and exceptions (z0 and z1 Core Variants)

The CPU exception sources are characterized in an integer array `os_CPU_vectors`, as for traditional variants but instead are implemented as a table of 16-byte aligned interrupt exception handler addresses. The IVPR register is set to be the first address of the `os_CPU_vectors` table and as such the table must be placed before the INTC vector table and within a common 64 Kbytes blocks of memory. A typical IVOR interrupt table is shown below:

```
  .section os_cpuvec,,c
  .alignn  4
  .global  os_CPU_vectors
os_CPU_vectors:
  .alignn 16
  e_b   Critical_Input                ; IVOR0
  .alignn 16
  e_b   Machine_Check                 ; IVOR1
  .alignn 16
  e_b   Data_Storage                  ; IVOR2
  .alignn 16
  e_b   Instruction_Storage           ; IVOR3
  .alignn 16
  e_b   Interrupt_exception_handler   ; IVOR4
  .alignn 16
  e_b   Alignment                     ; IVOR5
  .alignn 16
  e_b   Program                       ; IVOR6
  .alignn 16
  e_b   Floating_Point                ; IVOR7
  .alignn 16
  e_b   System_Call                   ; IVOR8
```

```
.alignn 16
e_b   Not_Used                          ; IVOR9
.alignn 16
e_b   Decrementer                       ; IVOR10
.alignn 16
e_b   Fixed_Interval_Timer              ; IVOR11
.alignn 16
e_b   Watchdog_Timer                    ; IVOR12
.alignn 16
e_b   Data_TLB_Error                    ; IVOR13
.alignn 16
e_b   Instruction_TLB_Error             ; IVOR14
.alignn 16
e_b   Debug                             ; IVOR15
```

In previous MPC55xx variants, for example, the MPC5534, the vector location IVOR 0 was defined as "Reserved". However, in more recent MPC55xx variants IVOR 0 has been redefined as "Critical Input" and in accordance with these developments it is now supported by version 5.0.1 and later of RTA-OSEK. It should be noted that IVOR 0 will require initializing in any user application because previous versions of RTA-OSEK only initialized from IVOR 1 onwards. A reference implementation can be found in the RTA-OSEK example application, function `write_IVORs()` in file `target.h` and shown above for the z0/z1 variants.

INTC interrupts and exceptions (Traditional Variants)

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 16-byte aligned interrupt exception handler addresses, one for each interrupt 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.

INTC interrupts and exceptions (z0 and z1 Core Variants)

INTC exception sources are characterized in an integer array `os_INTC_vectors`, as for traditional variants but instead of being 16-byte aligned interrupt exception handlers, they're 4-byte aligned. The array still requires aligning to a 64 Kbytes boundary and using the IVPR register to dictate the base address of the vector table. The table for the INTC vectors

must be placed 2 Kbytes above the CPU vectors, an example of which is shown below.

```
SECTIONS
{
        /* Now place the interrupt vectors */
        GROUP : {
           os_CPU_vectors                       : {}
           os_INTC_vectors (TEXT) ALIGN(2048) : {}
        } > vectors

        ...
}
```

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 MPC5533, MPC5534, MPC5561, MPC5565, MPC5566, MPC5567, MPC5514 (Z1 core only), MPC5516 (Z1 core only), SPC563M, MPC5674F and the MPC55xx VLE Generic. Further variants can be supported by contacting ETAS.

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 INTC Race Condition

A race condition between the modification of the INTC_CPR value and a triggering interrupt has been observed. For the race condition to occur the interrupt must trigger on the exact instruction that updates the value on the INTC_CPR. The result is that an implicit scheduling point is missed by RTA-OSEK. So if a task should be activated as a result of the interrupt involved in this race condition, then the task activation will occur at the next implicit scheduling point rather than before the interrupted task is resumed. This problem has been resolved in v5.0.0 by the addition of an interrupt nesting

counter `os_isr_count`. The counter can be addressed using the Small Data Area.

When the interrupt controller is configured to use hardware interrupt vectors the manipulation of this counter is handled by the RTA-OSEK kernel libraries. If the interrupt controller is configured to use software interrupt vectors then `os_isr_counter` must be incremented at the start of the interrupt handler before global interrupts are re-enabled (as demonstrated in the code for `os_mid_wrapper` in section 3.1.7). At the end of the interrupt handler `os_isr_counter` must be decremented (again as demonstrated in `os_mid_wrapper`).

Important: Failure to maintain the interrupt nesting counter `os_isr_counter` correctly in software vector mode may result in synchronization points being missed in an application.

3.1.12 OS_LIFO_LOAD

The RTA-OSEK Component for MPC55xxVLE/WindRiver supports the macro OS_LIFO_LOAD. It can be used by writers of common interrupt entry functions to push a value onto the INTC LIFO. Refer to the comments in `ostarget.h` for further details.

3.1.13 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.

3.1.14 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 by 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	Notes
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.

Register	Notes
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): 96

Timing

API max usage (bytes): 96

Extended

API max usage (bytes): 128

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 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	46	46	46	46	46	46
	ROM	148	148	152	232	232	236
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	220	220	224	304	304	308
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	276	276	280	360	360	364
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	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	82	82	82	82	82	82
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	106	106	106	106	106	106
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
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	70	70	70	70	70	70
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
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	n/a	n/a	n/a	n/a	n/a	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
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	102	102	102	102	102	102
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	106	106	106	106	106	106
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
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	70	70	70	70	70	70
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	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	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	114	114	114	114	114	114
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Alarm	RAM	12	12	12	12	12	12
	ROM	46	46	46	46	46	46
Counter	RAM	4	4	4	4	4	4
	ROM	110	110	110	110	110	110
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
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	70	70	70	70	70	70
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.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.

Variant	Description
Task	Resource is used only by tasks.

Standard

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	140	202	236	146	208	266
	NS		122	186	218	128	192	248
	KL	2	72	132	174	78	138	202
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	130	192	222	136	198	252
	SWH	1, 9	148	212	246	158	218	276
	NSL	8	130	192	222	136	198	252
	NSH	9	142	206	240	152	212	270
Schedule			98	98	114	98	98	114
GetTaskID			26	26	26	26	26	26
GetTaskState			84	84	84	98	98	98
EnableAllInterrupts			32	32	32	32	32	32
DisableAllInterrupts			40	40	40	40	40	40
ResumeAllInterrupts			46	46	46	46	46	46
SuspendAllInterrupts			58	58	58	58	58	58
ResumeOSInterrupts			40	40	40	40	40	40
SuspendOSInterrupts			60	60	60	60	60	60
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	80	80	80	80	80	80
	Combined	6	170	170	170	170	170	170
	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	194
	NS		n/a	n/a	n/a	94	94	168
	NS1i	10	n/a	n/a	n/a	64	n/a	n/a
	KL	2	n/a	n/a	n/a	60	60	134
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	50	50	50

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetEvent			n/a	n/a	n/a	10	10	10
WaitEvent	<default>		n/a	n/a	n/a	214	214	392
	fp	11	n/a	n/a	n/a	242	242	448
	1i	10	n/a	n/a	n/a	14	n/a	n/a
GetAlarmBase			66	66	66	66	66	66
GetAlarm			98	98	98	98	98	98
SetRelAlarm			430	430	430	430	430	430
SetAbsAlarm			456	456	456	456	456	456
CancelAlarm			92	92	92	92	92	92
InitCounter			60	60	60	60	60	60
GetCounterValue			78	78	78	78	78	78
GetScheduleTableStatus		34	68	86	86	68	86	86
NextScheduleTable		34	94	262	262	94	262	262
StartScheduleTable		34	132	186	186	132	186	186
StopScheduleTable		34	96	124	124	96	124	124
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		12	12	12	12	12	12
ScheduleTable expiry point	Final		48	48	48	48	48	48
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	38	38	38	38	38	38
Process container	Non-Yielding	33	26	26	26	26	26	26
osek_tick_alarm	<default>		80	80	80	80	80	80
	KL	2	34	34	34	34	34	34
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			230	230	230	230	230	230
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			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	74	74	74	172	172	172

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	172	172	172	172	172	172
GetMessageResource			56	56	56	56	56	56
ReleaseMessageResource			52	52	52	52	52	52
GetMessageStatus			54	54	54	54	54	54
SendMessage	SW CCCA	1, 14	92	92	92	218	218	218
	SW CCCB	1, 15	204	204	204	218	218	218
	NS CCCA	14	92	92	92	218	218	218
	NS CCCB	15	204	204	204	218	218	218
	KL CCCA	2, 14	64	64	64	184	184	184
	KL CCCB	2, 15	170	170	170	184	184	184
main_dispatch	NoHook	12	140	140	184	140	140	184
	Hook	13	174	174	216	174	174	216
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	250	300	n/a	250	300
	B2HF	25	n/a	258	308	n/a	258	308
	E1HI	26	n/a	n/a	n/a	332	332	398
	E1HF	27	n/a	n/a	n/a	340	340	406
	E2HI	28	n/a	n/a	n/a	n/a	n/a	398
	E2HF	29	n/a	n/a	n/a	n/a	n/a	406
ErrorHook support		16	60	60	60	60	60	60
	ServiceID	17	68	68	68	68	68	68
	Parameters	18	82	82	82	82	82	82
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	136	250	286	148	272	326
	NS		118	232	268	130	254	308
	KL	2	68	186	226	80	208	264
ChainTaskset	SWL	1, 8	126	252	288	132	268	322
	SWH	1, 9	168	282	318	174	298	352
	NSL	8	126	252	288	132	268	322
	NSH	9	162	276	312	168	292	346
GetTasksetRef			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	
MergeTaskset			60	60	60	60	60	60
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			60	60	60	60	60	60
TestSubTaskset			66	66	66	66	66	66
TestEquivalentTaskset			64	64	64	64	64	64
TickSchedule	SW	1	188	172	172	172	172	172
	NS		168	152	152	152	152	152
	KL	2	136	118	118	118	118	118
AdvanceSchedule	SW	1	180	158	158	158	158	158
	NS		160	138	138	138	138	138
	KL	2	124	106	106	106	106	106
StartSchedule			90	90	90	90	90	90
StopSchedule			76	76	76	76	76	76
GetScheduleStatus			102	102	102	102	102	102
GetScheduleValue			82	82	82	82	82	82
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			4	4	4	4	4	4
GetLargestExecutionTime			6	6	6	6	6	6
ResetLargestExecutionTime			4	4	4	4	4	4
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	140	202	236	146	208	266
	NS		122	186	218	128	192	248
	KL	2	72	132	174	78	138	202
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	130	192	222	136	198	252
	SWH	1, 9	148	212	246	158	218	276
	NSL	8	130	192	222	136	198	252
	NSH	9	142	206	240	152	212	270
Schedule			122	122	138	122	122	138
GetTaskID			26	26	26	26	26	26
GetTaskState			84	84	84	98	98	98
EnableAllInterrupts			32	32	32	32	32	32
DisableAllInterrupts			40	40	40	40	40	40
ResumeAllInterrupts			46	46	46	46	46	46
SuspendAllInterrupts			58	58	58	58	58	58
ResumeOSInterrupts			40	40	40	40	40	40
SuspendOSInterrupts			60	60	60	60	60	60
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	104	104	104	104	104	104
	Combined	6	216	216	216	216	216	216
	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	194
	NS		n/a	n/a	n/a	94	94	168
	NS1i	10	n/a	n/a	n/a	64	n/a	n/a
	KL	2	n/a	n/a	n/a	60	60	134
	KL1i	2, 10	n/a	n/a	n/a	20	n/a	n/a
ClearEvent			n/a	n/a	n/a	50	50	50
GetEvent			n/a	n/a	n/a	10	10	10
WaitEvent	<default>		n/a	n/a	n/a	282	282	450
	fp	11	n/a	n/a	n/a	310	310	506

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	102	n/a	n/a
GetAlarmBase			66	66	66	66	66	66
GetAlarm			98	98	98	98	98	98
SetRelAlarm			430	430	430	430	430	430
SetAbsAlarm			456	456	456	456	456	456
CancelAlarm			92	92	92	92	92	92
InitCounter			60	60	60	60	60	60
GetCounterValue			78	78	78	78	78	78
GetScheduleTableStatus		34	68	86	86	68	86	86
NextScheduleTable		34	94	262	262	94	262	262
StartScheduleTable		34	132	186	186	132	186	186
StopScheduleTable		34	96	124	124	96	124	124
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		12	12	12	12	12	12
ScheduleTable expiry point	Final		48	48	48	48	48	48
GetISRID		4	36	36	36	36	36	36
Process container	Yielding	32	38	38	38	38	38	38
Process container	Non-Yielding	33	26	26	26	26	26	26
osek_tick_alarm	<default>		80	80	80	80	80	80
	KL	2	34	34	34	34	34	34
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			276	276	276	276	276	276
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			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	74	74	74	172	172	172
	CCCB	15	172	172	172	172	172	172
GetMessageResource			56	56	56	56	56	56
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			54	54	54	54	54	54
SendMessage	SW CCCA	1, 14	92	92	92	218	218	218
	SW CCCB	1, 15	204	204	204	218	218	218
	NS CCCA	14	92	92	92	218	218	218
	NS CCCB	15	204	204	204	218	218	218
	KL CCCA	2, 14	64	64	64	184	184	184
	KL CCCB	2, 15	170	170	170	184	184	184
main_dispatch	NoHook	12	178	178	220	178	178	220
	Hook	13	212	212	256	212	212	256
sub_dispatch	B1LF	19	26	26	26	26	26	26
	B1HI	20	100	100	100	100	100	100
	B1HF	21	108	108	108	108	108	108
	B2LI	22	n/a	54	86	n/a	54	86
	B2LF	23	n/a	60	92	n/a	60	92
	B2HI	24	n/a	230	294	n/a	230	294
	B2HF	25	n/a	238	302	n/a	238	302
	E1HI	26	n/a	n/a	n/a	390	390	444
	E1HF	27	n/a	n/a	n/a	398	398	452
	E2HI	28	n/a	n/a	n/a	n/a	n/a	444
	E2HF	29	n/a	n/a	n/a	n/a	n/a	452
ErrorHook support		16	60	60	60	60	60	60
	ServiceID	17	68	68	68	68	68	68
	Parameters	18	82	82	82	82	82	82
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	136	250	286	148	272	326
	NS		118	232	268	130	254	308
	KL	2	68	186	226	80	208	264
ChainTaskset	SWL	1, 8	126	252	288	132	268	322
	SWH	1, 9	168	282	318	174	298	352
	NSL	8	126	252	288	132	268	322
	NSH	9	162	276	312	168	292	346
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			60	60	60	60	60	60
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			60	60	60	60	60	60

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
TestSubTaskset			66	66	66	66	66	66
TestEquivalentTaskset			64	64	64	64	64	64
TickSchedule	SW	1	188	172	172	172	172	172
	NS		168	152	152	152	152	152
	KL	2	136	118	118	118	118	118
AdvanceSchedule	SW	1	180	158	158	158	158	158
	NS		160	138	138	138	138	138
	KL	2	124	106	106	106	106	106
StartSchedule			90	90	90	90	90	90
StopSchedule			76	76	76	76	76	76
GetScheduleStatus			102	102	102	102	102	102
GetScheduleValue			82	82	82	82	82	82
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			122	122	122	122	122	122
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			12	12	12	12	12	12
GetStackOffset			18	18	18	18	18	18

Extended

Configuration Events Shared Task Priorities Multiple Task Activations			Application Uses					
			No			Yes		
			No		Yes	No		Yes
			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	242	306	340	248	312	364
	NS		300	364	400	306	370	424
	KL	2	164	232	268	170	238	288
TerminateTask	LExt	3	118	118	118	118	118	118

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	144	144	144	144	144	144
ChainTask	SWL	1, 8	274	338	370	280	344	398
	SWH	1, 9	306	368	398	312	374	422
	NSL	8	340	406	438	346	414	468
	NSH	9	366	428	460	372	434	486
Schedule			242	242	258	242	242	258
GetTaskID			46	46	46	46	46	46
GetTaskState			224	224	224	230	230	230
EnableAllInterrupts			52	52	52	52	52	52
DisableAllInterrupts			60	60	60	60	60	60
ResumeAllInterrupts			100	100	100	100	100	100
SuspendAllInterrupts			78	78	78	78	78	78
ResumeOSInterrupts			94	94	94	94	94	94
SuspendOSInterrupts			80	80	80	80	80	80
GetResource	Task	7	332	332	296	332	332	296
	Combined	6	310	310	310	310	310	310
	CLEx	3	270	270	270	270	270	270
ReleaseResource	Task	7	318	318	318	318	318	318
	Combined	6	426	426	426	426	426	426
	CLEx	3	276	276	276	276	276	276
SetEvent	SW	1	n/a	n/a	n/a	278	278	348
	NS		n/a	n/a	n/a	332	332	406
	NS1i	10	n/a	n/a	n/a	228	n/a	n/a
	KL	2	n/a	n/a	n/a	196	196	268
	KL1i	2, 10	n/a	n/a	n/a	162	n/a	n/a
ClearEvent			n/a	n/a	n/a	142	142	142
GetEvent			n/a	n/a	n/a	160	160	160
WaitEvent	<default>		n/a	n/a	n/a	374	374	538
	fp	11	n/a	n/a	n/a	402	402	594
	1i	10	n/a	n/a	n/a	204	n/a	n/a
GetAlarmBase			184	184	184	184	184	184
GetAlarm			172	172	172	172	172	172
SetRelAlarm			540	540	540	540	540	540
SetAbsAlarm			556	556	556	556	556	556
CancelAlarm			162	162	162	162	162	162
InitCounter			216	216	216	216	216	216
GetCounterValue			196	196	196	196	196	196

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	92	110	110	92	110	110
NextScheduleTable		34	114	282	282	114	282	282
StartScheduleTable		34	156	210	210	156	210	210
StopScheduleTable		34	116	144	144	116	144	144
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	14	14	14
ScheduleTable expiry point	Callback		4	4	4	4	4	4
ScheduleTable expiry point	Tick counter		12	12	12	12	12	12
ScheduleTable expiry point	Final		48	48	48	48	48	48
GetISRID		4	56	56	56	56	56	56
Process container	Yielding	32	38	38	38	38	38	38
Process container	Non-Yielding	33	26	26	26	26	26	26
osek_tick_alarm	<default>		118	118	118	118	118	118
	KL	2	34	34	34	34	34	34
osek_incr_counter			38	38	38	38	38	38
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			296	296	296	296	296	296
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			52	52	52	52	52	52
ReadFlag			38	38	38	38	38	38
ResetFlag			42	42	42	42	42	42
ReceiveMessage	CCCA	14	164	164	164	258	258	258
	CCCB	15	258	258	258	258	258	258
GetMessageResource			110	110	110	110	110	110
ReleaseMessageResource			112	112	112	112	112	112
GetMessageStatus			120	120	120	120	120	120
SendMessage	SW CCCA	1, 14	200	200	200	318	318	318
	SW CCCB	1, 15	304	304	304	318	318	318
	NS CCCA	14	200	200	200	318	318	318
	NS CCCB	15	304	304	304	318	318	318
	KL CCCA	2, 14	150	150	150	264	264	264
	KL CCCB	2, 15	250	250	250	264	264	264
main_dispatch	NoHook	12	178	178	220	178	178	220

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	212	212	256	212	212	256
sub_dispatch	B1LF	19	26	26	26	26	26	26
	B1HI	20	100	100	100	100	100	100
	B1HF	21	108	108	108	108	108	108
	B2LI	22	n/a	54	86	n/a	54	86
	B2LF	23	n/a	60	92	n/a	60	92
	B2HI	24	n/a	230	294	n/a	230	294
	B2HF	25	n/a	238	302	n/a	238	302
	E1HI	26	n/a	n/a	n/a	390	390	444
	E1HF	27	n/a	n/a	n/a	398	398	452
	E2HI	28	n/a	n/a	n/a	n/a	n/a	444
	E2HF	29	n/a	n/a	n/a	n/a	n/a	452
ErrorHook support		16	128	128	128	128	128	128
	ServiceID	17	138	138	138	138	138	138
	Parameters	18	156	156	156	156	156	156
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	312	358	402	328	380	454
	NS		366	412	458	382	432	502
	KL	2	230	270	310	242	292	348
ChainTaskset	SWL	1, 8	352	398	458	360	412	494
	SWH	1, 9	402	454	508	410	470	552
	NSL	8	414	472	522	422	488	562
	NSH	9	460	526	564	468	544	610
GetTasksetRef			140	140	140	140	140	140
MergeTaskset			256	256	256	256	256	256
AssignTaskset			168	168	168	168	168	168
RemoveTaskset			256	256	256	256	256	256
TestSubTaskset			272	272	272	272	272	272
TestEquivalentTaskset			270	270	270	270	270	270
TickSchedule	SW	1	300	272	272	272	272	272
	NS		352	342	342	342	342	342
	KL	2	236	210	210	210	210	210
AdvanceSchedule	SW	1	300	274	274	274	274	274
	NS		354	340	340	340	340	340
	KL	2	244	216	216	216	216	216

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			234	234	234	234	234	234
StopSchedule			190	190	190	190	190	190
GetScheduleStatus			218	218	218	218	218	218
GetScheduleValue			176	176	176	176	176	176
GetScheduleNext			82	82	82	82	82	82
SetScheduleNext			172	172	172	172	172	172
GetArrivalpointDelay			132	132	132	132	132	132
SetArrivalpointDelay			136	136	136	136	136	136
GetArrivalpointTasksetRef			120	120	120	120	120	120
GetArrivalpointNext			132	132	132	132	132	132
SetArrivalpointNext			168	168	168	168	168	168
TestArrivalpointWritable			142	142	142	142	142	142
GetExecutionTime			170	170	170	170	170	170
GetLargestExecutionTime			106	106	106	106	106	106
ResetLargestExecutionTime			100	100	100	100	100	100
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

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	142	203	260	152	196	253
	NS	149	202	241	143	183	242
	KL	89	146	205	92	127	195
TerminateTask	LExt	0	0	0	0	0	0
	H	262	254	263	254	264	256
ChainTask	SWL	416	486	591	471	528	635
	SWH	539	596	708	586	646	767
	NSL	418	485	590	471	538	643
	NSH	535	591	705	589	639	749
Schedule	SW	140	140	149	137	140	148
GetTaskID		36	36	36	38	37	36
GetTaskState		118	119	113	130	124	124
EnableAllInterrupts		50	50	51	51	53	51
DisableAllInterrupts		54	54	53	56	54	53
ResumeAllInterrupts		63	63	63	63	63	62
SuspendAllInterrupts		67	67	68	69	67	69
ResumeOSInterrupts		63	63	63	63	63	62
SuspendOSInterrupts		67	67	68	69	67	69
GetResource	Task	56	56	56	57	59	58
	Combined	105	105	99	92	98	92
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	117	117	117	110	116	121
	Combined	173	173	173	168	176	178
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	140	135	147
	NS	n/a	n/a	n/a	122	132	136
	KL	n/a	n/a	n/a	84	87	90
ClearEvent		n/a	n/a	n/a	80	80	80
GetEvent		n/a	n/a	n/a	40	40	40
WaitEvent	<default>	n/a	n/a	n/a	715	724	781
	fp	n/a	n/a	n/a	728	742	803
GetAlarmBase		120	115	112	119	112	111
GetAlarm		133	135	140	132	139	139

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		184	169	175	184	171	178
SetAbsAlarm		191	199	196	191	199	196
CancelAlarm		108	116	114	107	115	113
InitCounter		84	84	90	89	85	86
GetCounterValue		114	114	112	109	116	108
osek_tick_alarm	<default>	106	106	103	107	104	104
	KL	51	51	52	49	48	50
osek_incr_counter		27	27	27	27	27	27
GetActiveApplicationMode		15	15	15	16	16	16
StartOS		1282	1333	1337	1309	1255	1252
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	68	69	69	74	71	68
InitCOM		23	23	23	19	21	20
CloseCOM		23	22	22	20	19	20
StartCOM		73	73	73	113	111	109
StopCOM		34	34	34	32	29	29
ReadFlag		n/a	n/a	n/a	22	22	22
ResetFlag		n/a	n/a	n/a	20	20	20
ReceiveMessage		95	97	97	349	345	346
GetMessageResource		n/a	n/a	n/a	153	153	156
ReleaseMessageResource		n/a	n/a	n/a	194	187	188
GetMessageStatus		n/a	n/a	n/a	65	72	73
SendMessage	SW	267	329	389	517	564	629
	NS	277	326	363	512	553	613
	KL	175	234	287	423	476	532
ActivateTaskset	SW	123	705	787	127	635	805
	NS	110	655	774	112	843	844
	KL	58	641	729	58	574	713
	SW2	123	705	787	127	635	805
	NS2	110	655	774	112	843	844
	KL2	58	641	729	58	574	713
ChainTaskset	SWL	395	951	1059	449	1001	1147
	SWH	525	1046	1157	570	1088	1276
	NSL	399	952	1123	445	970	1147
	NSH	520	1073	1156	570	1086	1306
GetTasksetRef		34	34	34	34	33	33

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		91	87	93	91	94	94
AssignTaskset		31	32	31	33	32	32
RemoveTaskset		83	87	83	90	84	84
TestSubTaskset		95	90	97	94	98	98
TestEquivalentTaskset		89	93	89	96	90	90
TickSchedule	SW	215	848	938	263	810	936
	NS	202	819	908	248	779	910
	KL	163	792	882	203	750	880
	SW2	215	848	938	263	788	925
	NS2	202	818	908	248	757	899
	KL2	163	792	882	203	728	869
AdvanceSchedule	SW	194	812	908	227	770	906
	NS	166	797	888	216	753	886
	KL	130	755	846	171	711	840
	SW2	194	812	908	227	748	895
	NS2	166	797	888	216	731	875
	KL2	130	755	845	171	690	829
StartSchedule		150	150	150	141	147	148
StopSchedule		129	129	129	129	125	128
GetScheduleStatus		142	142	142	138	141	143
GetScheduleValue		129	129	129	121	123	124
GetScheduleNext		37	37	37	39	39	40
SetScheduleNext		36	36	36	35	35	34
GetArrivalpointDelay		34	34	34	34	34	33
SetArrivalpointDelay		27	27	27	29	26	26
GetArrivalpointTasksetRef		24	24	24	25	25	25
GetArrivalpointNext		31	31	31	33	32	32
SetArrivalpointNext		26	26	26	28	25	25
TestArrivalpointWritable		45	38	38	38	43	45
GetExecutionTime		24	25	25	22	21	21
GetLargestExecutionTime		30	30	30	31	31	31
ResetLargestExecutionTime		25	23	23	24	25	26
GetStackOffset		28	28	28	26	26	26

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	160	207	260	164	196	256
	NS	134	199	241	144	183	243
	KL	88	152	205	94	127	204
TerminateTask	LExt	0	0	0	0	0	0
	H	512	497	504	505	504	508
ChainTask	SWL	703	784	868	785	817	915
	SWH	807	890	975	897	927	1029
	NSL	710	783	868	787	826	924
	NSH	817	879	973	893	921	1027
Schedule	SW	140	140	146	143	140	151
GetTaskID		36	37	36	37	37	37
GetTaskState		119	119	113	125	124	130
EnableAllInterrupts		48	50	53	58	53	58
DisableAllInterrupts		50	54	50	54	57	54
ResumeAllInterrupts		59	63	62	63	66	63
SuspendAllInterrupts		66	67	66	67	70	67
ResumeOSInterrupts		59	63	62	63	66	63
SuspendOSInterrupts		66	67	66	67	70	67
GetResource	Task	54	56	58	58	58	62
	Combined	96	105	97	99	101	99
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	116	115	113	109	112	116
	Combined	168	169	176	176	171	175
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	134	134	143
	NS	n/a	n/a	n/a	131	131	135
	KL	n/a	n/a	n/a	86	86	89
ClearEvent		n/a	n/a	n/a	81	80	83
GetEvent		n/a	n/a	n/a	41	40	40
WaitEvent	<default>	n/a	n/a	n/a	950	947	973
	fp	n/a	n/a	n/a	966	961	987
GetAlarmBase		113	112	112	112	113	121
GetAlarm		140	136	140	139	139	133

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		171	171	178	171	171	177
SetAbsAlarm		199	195	196	199	199	190
CancelAlarm		116	113	114	115	115	113
InitCounter		87	86	90	82	85	85
GetCounterValue		110	117	112	113	116	116
osek_tick_alarm	<default>	103	105	103	104	104	107
	KL	50	51	52	48	48	49
osek_incr_counter		27	27	27	27	27	27
GetActiveApplicationMode		15	15	15	16	16	16
StartOS		3584	3682	3806	3578	3573	3813
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	74	68	71	69	68	74
InitCOM		23	22	22	19	19	19
CloseCOM		22	23	22	20	20	19
StartCOM		77	76	76	117	119	119
StopCOM		34	37	34	29	32	29
ReadFlag		n/a	n/a	n/a	22	22	22
ResetFlag		n/a	n/a	n/a	20	20	20
ReceiveMessage		97	105	104	349	350	352
GetMessageResource		n/a	n/a	n/a	153	155	152
ReleaseMessageResource		n/a	n/a	n/a	188	186	196
GetMessageStatus		n/a	n/a	n/a	66	73	66
SendMessage	SW	289	334	389	532	566	634
	NS	256	320	363	514	555	615
	KL	170	240	287	443	478	540
ActivateTaskset	SW	116	651	787	129	635	809
	NS	104	628	774	105	843	759
	KL	57	583	729	60	574	741
	SW2	116	651	787	129	635	809
	NS2	104	628	774	105	843	759
	KL2	57	583	729	60	574	741
ChainTaskset	SWL	691	1276	1336	750	1289	1501
	SWH	801	1358	1424	876	1369	1577
	NSL	689	1465	1400	758	1259	1501
	NSH	801	1333	1424	879	1367	1634
GetTasksetRef		35	34	34	33	33	33

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		90	87	93	92	94	88
AssignTaskset		32	32	31	32	32	33
RemoveTaskset		89	90	83	84	84	88
TestSubTaskset		93	90	97	96	98	91
TestEquivalentTaskset		95	96	89	90	90	94
TickSchedule	SW	210	788	938	267	803	971
	NS	197	773	923	252	788	937
	KL	167	731	875	200	743	908
	SW2	210	787	938	267	781	960
	NS2	197	773	923	252	766	926
	KL2	167	730	875	200	721	897
AdvanceSchedule	SW	184	765	908	232	770	923
	NS	177	742	888	215	753	912
	KL	138	701	845	173	711	870
	SW2	184	765	908	232	748	912
	NS2	177	742	888	215	731	901
	KL2	138	701	845	173	689	859
StartSchedule		147	140	140	138	141	151
StopSchedule		128	129	128	128	129	127
GetScheduleStatus		145	140	139	141	138	143
GetScheduleValue		127	124	127	127	121	127
GetScheduleNext		38	37	37	40	39	39
SetScheduleNext		36	37	36	34	35	34
GetArrivalpointDelay		35	34	34	33	33	34
SetArrivalpointDelay		27	27	27	26	26	29
GetArrivalpointTasksetRef		27	24	24	25	25	25
GetArrivalpointNext		32	31	31	32	32	33
SetArrivalpointNext		29	26	26	25	25	28
TestArrivalpointWritable		46	45	43	38	45	38
GetExecutionTime		156	152	149	145	149	148
GetLargestExecutionTime		51	50	49	50	51	51
ResetLargestExecutionTime		40	41	41	43	42	42
GetStackOffset		28	28	28	26	26	26

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	560	589	661	567	614	681
	NS	621	639	727	632	679	733
	KL	493	519	595	495	551	614
TerminateTask	LExt	560	561	565	574	570	566
	H	655	653	657	660	661	656
ChainTask	SWL	1194	1265	1382	1273	1349	1449
	SWH	1316	1350	1467	1381	1451	1534
	NSL	1282	1336	1448	1350	1425	1519
	NSH	1374	1417	1539	1451	1512	1590
Schedule	SW	216	226	231	224	227	225
GetTaskID		45	44	44	45	46	44
GetTaskState		560	538	566	573	593	600
EnableAllInterrupts		59	56	57	60	60	62
DisableAllInterrupts		60	63	60	64	68	66
ResumeAllInterrupts		84	87	86	87	90	92
SuspendAllInterrupts		76	73	73	75	77	79
ResumeOSInterrupts		84	87	86	87	90	92
SuspendOSInterrupts		76	73	73	75	77	79
GetResource	Task	919	912	422	1001	1016	528
	Combined	389	387	399	483	497	489
	CLEx	433	415	442	511	525	525
ReleaseResource	Task	414	409	418	502	515	515
	Combined	432	424	428	505	525	525
	CLEx	409	404	409	497	507	507
SetEvent	SW	n/a	n/a	n/a	606	624	624
	NS	n/a	n/a	n/a	621	652	648
	KL	n/a	n/a	n/a	537	570	559
ClearEvent		n/a	n/a	n/a	145	140	145
GetEvent		n/a	n/a	n/a	499	533	521
WaitEvent	<default>	n/a	n/a	n/a	1065	1066	1098
	fp	n/a	n/a	n/a	1064	1078	1121
GetAlarmBase		366	349	392	364	374	403
GetAlarm		376	360	403	375	386	415

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		444	432	471	443	461	490
SetAbsAlarm		449	436	476	445	463	492
CancelAlarm		351	330	378	352	364	393
InitCounter		578	603	620	600	619	628
GetCounterValue		332	319	325	334	338	346
osek_tick_alarm	<default>	139	142	139	139	138	138
	KL	50	50	50	49	50	50
osek_incr_counter		25	25	25	24	24	24
GetActiveApplicationMode		14	14	14	15	15	15
StartOS		3893	3813	3700	3700	3698	3698
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	73	75	70	70	73	73
InitCOM		21	20	20	24	23	23
CloseCOM		22	22	21	24	22	22
StartCOM		88	92	88	138	138	138
StopCOM		48	43	43	55	48	48
ReadFlag		n/a	n/a	n/a	48	48	48
ResetFlag		n/a	n/a	n/a	44	44	44
ReceiveMessage		284	280	275	524	525	525
GetMessageResource		n/a	n/a	n/a	714	731	735
ReleaseMessageResource		n/a	n/a	n/a	700	739	731
GetMessageStatus		n/a	n/a	n/a	238	229	242
SendMessage	SW	860	884	960	1105	1159	1226
	NS	921	933	1018	1167	1221	1275
	KL	745	766	856	980	1041	1104
ActivateTaskset	SW	724	1435	1363	730	1226	1378
	NS	769	1351	1428	788	1319	1501
	KL	651	1247	1363	623	1354	1428
	SW2	724	1435	1363	730	1226	1378
	NS2	769	1351	1428	788	1319	1501
	KL2	651	1247	1363	623	1354	1428
ChainTaskset	SWL	1368	1953	2237	1471	1995	2224
	SWH	1553	2138	2251	1542	2112	2366
	NSL	1420	2101	2239	1534	2146	2246
	NSH	1518	2248	2335	1653	2127	2357
GetTasksetRef		465	439	466	469	498	492

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		187	200	202	194	196	203
AssignTaskset		114	107	111	112	105	109
RemoveTaskset		191	189	190	183	196	190
TestSubTaskset		191	190	195	186	183	193
TestEquivalentTaskset		185	188	191	184	185	191
TickSchedule	SW	298	1528	1635	886	1650	1719
	NS	371	1577	1696	945	1703	1772
	KL	221	1444	1577	829	1594	1663
	SW2	298	1528	1634	886	1618	1693
	NS2	371	1577	1695	945	1671	1746
	KL2	221	1444	1577	829	1562	1637
AdvanceSchedule	SW	276	1492	1618	869	1623	1692
	NS	332	1552	1670	921	1687	1756
	KL	216	1418	1542	798	1565	1634
	SW2	276	1492	1619	869	1591	1666
	NS2	332	1552	1671	921	1655	1730
	KL2	216	1418	1543	798	1533	1608
StartSchedule		246	239	248	248	244	244
StopSchedule		183	186	175	173	178	178
GetScheduleStatus		185	189	193	193	195	195
GetScheduleValue		181	178	184	187	182	182
GetScheduleNext		72	72	75	74	73	73
SetScheduleNext		137	117	130	130	116	116
GetArrivalpointDelay		88	95	91	90	88	88
SetArrivalpointDelay		119	103	114	113	109	109
GetArrivalpointTasksetRef		72	73	72	73	78	78
GetArrivalpointNext		85	80	80	78	78	78
SetArrivalpointNext		137	130	130	130	126	126
TestArrivalpointWritable		83	87	88	88	85	85
GetExecutionTime		186	190	184	188	186	186
GetLargestExecutionTime		447	414	448	446	470	470
ResetLargestExecutionTime		431	396	429	430	450	450
GetStackOffset		26	26	26	26	26	26

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	72	72	72	72	72	72
	Cat 2	118	226	226	216	215	219

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	72	72	72	72	72	72
	Cat 2	309	407	402	402	400	398

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	72	72	72	72	72	72
	Cat 2	301	404	402	403	399	399

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.

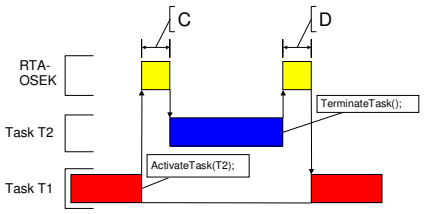

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

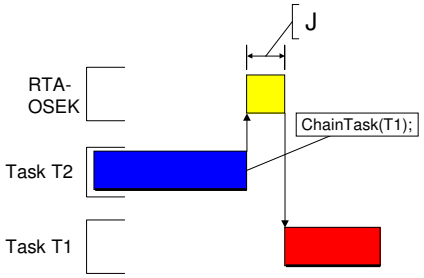

Figure 2: Task Chaining

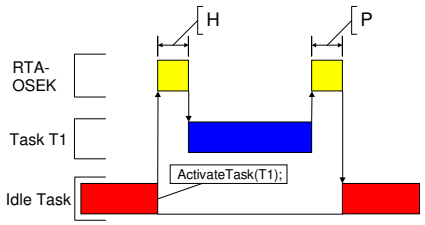

Figure 3: Task Activation from Idle Task

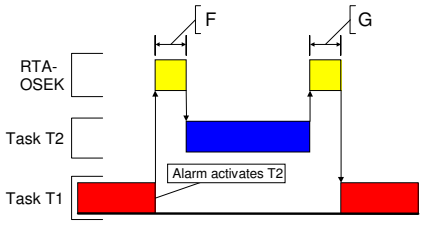

Figure 4: Task Activation from an Alarm

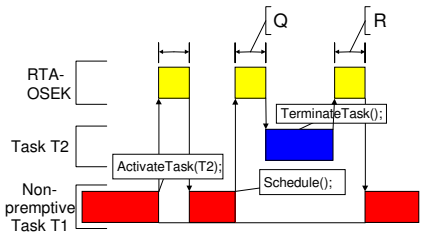

Figure 5: Non-Premptive Task Calls Schedule()

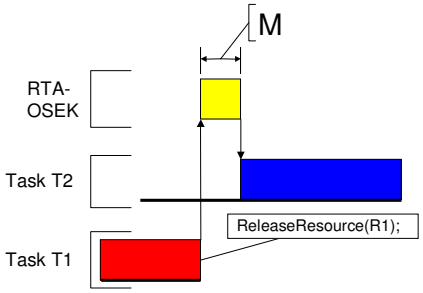

Figure 6: Blocked Task Activated by ReleaseResource()

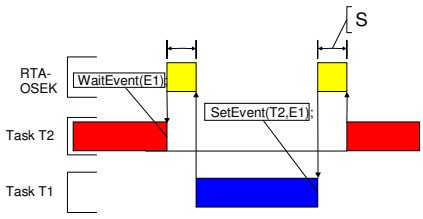

Figure 7: Waiting Task Activated by SetEvent()

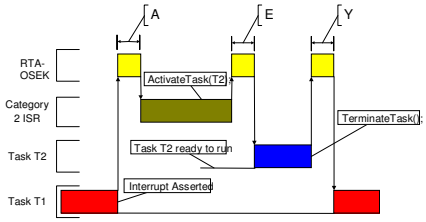

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	128	182	214	121	197	213
Figure 1: D	Heavy, Basic/Extended	261	308	342	322	327	347
ChainTask	Light, Basic	293	373	476	284	387	491
Figure 2: J	Heavy, Basic/Extended	701	815	959	748	849	982
Pre-emption	Light, Basic	219	300	427	225	308	444
Figure 1: C	Heavy, Basic/Extended	355	426	555	423	469	600
From idle task	Light, Basic	218	299	426	224	307	443
Figure 3: H	Heavy, Basic/Extended	354	425	554	422	468	599
Triggered by alarm	Light, Basic	392	472	593	399	477	612
Figure 4: F	Heavy, Basic/Extended	529	599	722	595	638	768
Schedule	Light, Basic	191	211	295	187	215	297
Figure 5: Q	Heavy, Basic/Extended	327	337	423	385	391	476
Release resource	Light, Basic	196	216	290	197	224	295
Figure 6: M	Heavy, Basic/Extended	332	342	418	395	400	474
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	727	736	900
From category 2 ISR	Light, Basic	200	312	389	291	319	395
Figure 8: E	Heavy, Basic/Extended	336	438	517	489	495	574

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	379	415	452	379	422	453
Figure 1: D	Heavy, Basic/Extended	511	530	572	549	545	575
ChainTask	Light, Basic	593	668	758	599	674	772
Figure 2: J	Heavy, Basic/Extended	1224	1327	1460	1287	1347	1482
Pre-emption	Light, Basic	384	443	551	387	449	565
Figure 1: C	Heavy, Basic/Extended	507	572	675	584	611	722
From idle task	Light, Basic	383	442	550	386	448	564
Figure 3: H	Heavy, Basic/Extended	506	571	674	583	610	721
Triggered by alarm	Light, Basic	552	615	717	555	619	739
Figure 4: F	Heavy, Basic/Extended	676	746	843	751	780	895
Schedule	Light, Basic	350	350	419	347	356	421
Figure 5: Q	Heavy, Basic/Extended	473	479	543	544	533	594
Release resource	Light, Basic	355	355	418	354	364	424
Figure 6: M	Heavy, Basic/Extended	478	484	542	551	541	597
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	846	835	985
From category 2 ISR	Light, Basic	618	715	776	711	723	781
Figure 8: E	Heavy, Basic/Extended	733	843	899	906	899	946

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	560	602	638	564	601	626
Figure 1: D	Heavy, Basic/Extended	656	689	722	703	701	719
ChainTask	Light, Basic	1082	1155	1269	1093	1206	1295
Figure 2: J	Heavy, Basic/Extended	1882	1956	2096	1925	2020	2130
Pre-emption	Light, Basic	757	802	928	763	846	971
Figure 1: C	Heavy, Basic/Extended	882	925	1054	955	1002	1124
From idle task	Light, Basic	758	803	929	764	847	972
Figure 3: H	Heavy, Basic/Extended	883	926	1055	956	1003	1125
Triggered by alarm	Light, Basic	965	1011	1134	972	1057	1180
Figure 4: F	Heavy, Basic/Extended	1090	1136	1262	1164	1213	1333
Schedule	Light, Basic	390	415	481	393	413	477
Figure 5: Q	Heavy, Basic/Extended	515	538	607	585	591	650
Release resource	Light, Basic	612	619	686	690	730	791
Figure 6: M	Heavy, Basic/Extended	737	742	812	882	908	964
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1270	1303	1454
From category 2 ISR	Light, Basic	643	753	811	732	741	802
Figure 8: E	Heavy, Basic/Extended	762	875	929	923	918	974

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	160	176	n/a	160	176
BCC2 lightweight, floating-point	n/a	160	176	n/a	160	176
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	288	288	272
ECC1 heavyweight, floating-point	n/a	n/a	n/a	288	288	272
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	272
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	272
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	176	176	n/a	176	176
BCC2 lightweight, floating-point	n/a	176	176	n/a	176	176
BCC2 heavyweight, integer	n/a	272	272	n/a	272	272
BCC2 heavyweight, floating-point	n/a	272	272	n/a	272	272
ECC1 heavyweight, integer	n/a	n/a	n/a	272	272	272
ECC1 heavyweight, floating-point	n/a	n/a	n/a	272	272	272
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	272
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	272

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	160	160	160	160	160	160
BCC1 lightweight, floating-point	176	176	176	176	176	176
BCC1 heavyweight, integer	272	272	272	272	272	272
BCC1 heavyweight, floating-point	272	272	272	272	272	272
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	288	288	n/a	288	288
BCC2 heavyweight, floating-point	n/a	288	288	n/a	288	288
ECC1 heavyweight, integer	n/a	n/a	n/a	320	320	288
ECC1 heavyweight, floating-point	n/a	n/a	n/a	320	320	288
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	288
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	288
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	160	160	176	160	160	176
BCC1 lightweight, floating-point	176	176	192	176	176	192
BCC1 heavyweight, integer	272	272	288	272	272	288
BCC1 heavyweight, floating-point	272	272	288	272	272	288
BCC2 lightweight, integer	n/a	192	208	n/a	192	208
BCC2 lightweight, floating-point	n/a	192	208	n/a	192	208
BCC2 heavyweight, integer	n/a	288	304	n/a	288	304
BCC2 heavyweight, floating-point	n/a	288	304	n/a	288	304
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

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	160	160	160	160	160	160
BCC1 lightweight, floating-point	176	176	176	176	176	176
BCC1 heavyweight, integer	272	272	272	272	272	272
BCC1 heavyweight, floating-point	272	272	272	272	272	272
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	288	288	n/a	288	288
BCC2 heavyweight, floating-point	n/a	288	288	n/a	288	288
ECC1 heavyweight, integer	n/a	n/a	n/a	320	320	288
ECC1 heavyweight, floating-point	n/a	n/a	n/a	320	320	288
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	288
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	288
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	160	160	176	160	160	176
BCC1 lightweight, floating-point	176	176	192	176	176	192
BCC1 heavyweight, integer	272	272	288	272	272	288
BCC1 heavyweight, floating-point	272	272	288	272	272	288
BCC2 lightweight, integer	n/a	192	208	n/a	192	208
BCC2 lightweight, floating-point	n/a	192	208	n/a	192	208
BCC2 heavyweight, integer	n/a	288	304	n/a	288	304
BCC2 heavyweight, floating-point	n/a	288	304	n/a	288	304
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

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for the MPC55xxVLE/WindRiver supports two variations of the OSEK interrupt handling API calls. In addition to the API calls contained within the RTA-OSEK run-time libraries, inline versions are also supported. Using these inline versions will result in faster code and a reduced Context Save Area usage. The inline versions of these API calls are all have the "os" prefix.

The inline API calls are restricted to the standard build applications that do not use RTA-TRACE. Inline calls contained within application code for other configurations will be automatically substituted with calls to the library API during compilation.

To take advantage of the inline versions in application code the substitutions in the following table should be used:

Library API call	Inline API call
`DisableAllInterrupts()`	`osDisableAllInterrupts()`
`EnableAllInterrupts()`	`osEnableAllInterrupts()`
`SuspendOSInterrupts()`	`osSuspendOSInterrupts()`
`ResumeOSInterrupts()`	`osResumeOSInterrupts()`
`SuspendAllInterrupts()`	`osSuspendAllInterrupts()`
`ResumeAllInterrupts()`	`osResumeAllInterrupts()`

6 Version 5.0.1

6.1 Variants

The supported targets in the RTA-OSEK GUI have been extended from the MPC5534, MPC5561, MPC5565, MPC5566, MPC5567, MPC5514 (Z1 core only), MPC5516 (Z1 core only) and generic MPC55xxVLE to additionally include the MPC5533, SPC563M and MPC5674F.

6.2 Kernel Linking

If an attempt to link a v5.0.1 OIL file with a v5.00 kernel library is made, a linker error will be seen. This is due to incompatible difference between v5.00 and v5.01 kernels.

6.3 Issues with Compiler Version 5.8.0.0

In order to guarantee that instructions inserted via assembler macros are not optimized away, as previously observed with the WindRiver version 5.6.1.0, 5.7.0.0 and 5.7.0.0 (plus Diab5700-1_wind00175250 patch) compilers, the `volatile` statement has been incorporated into the kernel where applicable and should also be used in application code.

Important: ETAS strongly recommend the use of the `volatile` statement for assembler macros in application code.

6.4 Issues with Compiler Option –Xnested-interrupts

The compiler option `-Xnested-interrupts` is used to instruct the compiler to also save and restore the `SRR0/SRR1` registers during a Category 1 interrupt, i.e. those interrupts declared with the `__interrupt__` keyword. However, as a side effect of the option the compiler will also clear the `INTC_EOIR` at the end of the interrupt. Therefore, it is not necessary to additionally provide the code to clear the `INTC_EOIR`.

7 Version 5.0.0

7.1 Variants

The supported targets in the RTA-OSEK GUI have been extended from the MPC5534, MPC5565, MPC5567 and generic MPC55xx to additionally include the MPC5561, MPC5566, MPC5514 (Z1 core only) and MPC5516 (Z1 core only).

7.2 INTC Race Condition

A race condition between the modification of the INTC_CPR value and a triggering interrupt has been observed. For the race condition to occur the interrupt must trigger on the exact instruction that updates the value on the INTC_CPR. The result is that an implicit scheduling point is missed by RTA-OSEK. If tasks should be activated as a result of the interrupt in question the activation will occur at the next implicit scheduling point rather than at the end of the interrupt. If an application configures the INTC in software vector mode then the handled function must be modified along the guidelines section 3.1.7.

7.3 IVOR0 CPU Interrupt

In previous MPC55xx variants, for example, the MPC5534 the vector location IVOR 0 was defined as "Reserved". However, in more recent MPC55xx variants IVOR 0 has been redefined as "Critical Input" and in accordance with these developments it is now supported by version 5.0.0 of RTA-OSEK. It should be noted that IVOR 0 will require initializing in any user application because previous versions of RTA-OSEK only initialized from IVOR 1 onwards.

7.4 Compiler Incompatibility

Please note that because of a modification in which pragmas are used to override section names, there exists and incompatibility between version 5.3 and 5.4 onward compilers.

7.5 Linker Files

Linker command files (.dld) must be modified from the previous release to add the renamed stack identifier (os_SP_INIT), and also to add the new memory sections (os_pidf, os_trace_ram, os_pir2 and os_pnir2).

8 Compatibility with Pre-v5 Kernels

8.1 Updating the application version

To convert an existing v3.x OIL configuration file to v5.0.1, load the file into the v5 RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.01. When the OIL configuration file is saved it will then use the v5.0.1 format and the v5.01 kernel libraries. This process can be reversed to move back to earlier kernel versions.

8.2 Supporting software vector mode

To support software vector mode in the interrupt controller an additional variable `os_isr_counter` must be maintained in the interrupt handler. Details on correct handling of this variable can be found in section 3.1.11

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.