RTA-OSEK

Binding Manual: MPC55xxVLE/Metrowerks

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: RM00098-001

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 7
1.1 Who Should Read this Guide? 7
1.2 Conventions 7
2 Toolchain Issues 9
2.1 Compiler 9
2.2 Assembler 10
2.3 Linker/Locator 10
2.4 Debugger 11
3 Target Hardware Issues 13
3.1 Interrupts 13
3.1.1 Interrupt Levels 13
3.1.2 Interrupt Vectors 13
3.1.3 Category 1 Handlers 14
3.1.4 Category 2 Handlers 14

3.1.5 Vector Table Issues 14
3.1.6 Processor Mode 14
3.1.7 INTC vector mode 15
3.1.8 INTC and CPU Vector Offset Mapping 19
3.1.9 Number of supported INTC vectors 21
3.1.10 INTC PSR register initialization 21
3.1.11 INTC Race Condition 22
3.1.12 OS_LIFO_LOAD 22
3.1.13 Default Interrupt 22
3.1.14 Addition of user ISR timing hooks 23
3.2 Register Settings 24
3.3 Stack Usage 25
3.3.1 Number of Stacks 25
3.3.2 Stack Usage within API Calls 25
3.3.3 Stack discipline 26
3.4 Floating point 26
4 Parameters of Implementation 29
4.1 Functionality 29
4.2 Hardware Resources 30
4.2.1 ROM and RAM Overheads 30
4.2.2 ROM and RAM for OSEK OS Objects 31
4.2.3 Size of Linkable Modules 36
4.2.4 Reserved Hardware Resources 50
4.3 Performance 50
4.3.1 Execution Times for RTA-OSEK API Calls 50
4.3.2 OS Start-up Time 60
4.3.3 Interrupt Latencies 60
4.3.4 Task Switching Times 61
4.4 Configuration of Run-time Context 64
5 Inline Interrupt Control API Calls 68
6 Version 5.0.0 69
6.1 Variants 69
6.2 OS_FIFO_LOAD() 69

6.3 CPU Vector Generation Setting 69

1 About this Guide

This guide provides target-specific information for the MPC55xxVLE/Metrowerks 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/Metrowerks supports the single flat memory model supported by the Metrowerks toolchain. This toolchain supports the Embedded Application Binary Interface, EABI. However, it should be noted that the Metrowerks startup code uses 8-bytes of stack rather than the customary 16-byte multiple.

2.1 Compiler

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

Vendor	Metrowerks
Compiler	Freescale C/C++ Compiler for Embedded PowerPC
Version	2.6

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

Option	**Description**
`-c`	compile to object without linking

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**
`-sym off`	Don't generate debugging information
`-c`	Don't link
`-O2,p`	Optimize for speed
`-vle`	Generate VLE code
`-ppc_asm_to_vle`	Allow Book E asm to be used and converted to VLE
`-nodefaults`	Treats #include <> the same as #include " "
`-ir %CBASE_INC%`	Appends a recursive path to the current #include list
`-I-`	Search system paths first
`-sdata 0`	Limits the size of the largest objects in the small data section

Option	Description
-sdata2 0	Limits the size of the largest objects in the small constant data section

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

Option	Description
-sym on|full|dwarf-1|dwarf-2	Debugging must be disabled.

To support the use of multiple CPU configurations the –proc command line option can be set if desired e.g. –proc Zen.

2.2 Assembler

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

Vendor	Metrowerks
Assembler	Freescale Assembler for Embedded PowerPC
Version	2.6

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

Option	Description
-c	assemble to object without linking

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
-c	assemble to object without linking

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. 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	INTC vector table if generated by RTA-OSEK (Should be aligned on a 64KByte boundary)
`os_cpuvec`	ROM	CPU vector table (Should be aligned on a 2KByte boundary)
`os_text`	ROM	RTA-OSEK code section. This section is typically empty in this release.
`os_text_vle`	ROM	RTA-OSEK VLE 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_trace_ram`	RAM	RTA-TRACE buffer. RTA-TRACE buffer. Can be located in 'far' RAM. Does not need to be initialized.
`os_pur`	RAM	RTA-OSEK uninitialized data. Zeroed during StartOS().
`os_pur2`	RAM	RTA-OSEK uninitialized data. Must be zeroed during C-startup.

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.

As is customary with some compilers, if a variable is declared but not referenced within the code, the compiler will remove it and this is known as dead-stripping. In order to prevent this occurring to ORTI variables declared in `osekdefs.c` the `FORCEFILES{}` command is required to be used in the Linker Command File (LCF), for example, `FORCEFILES{osekdefs.o}`.

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for MPC55xxVLE/Metrowerks. 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 *MPC5668G 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 Metrowerks C compiler can generate appropriate interrupt handling code for a C function decorated with the `__declspec(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

  e_stwu    r1,-16(r1)          ; Allocate some
                                ; stack
  se_stw    r3,8(r1)            ; Store r3
  se_mfctr  r3                  ; Store the ctr
  se_stw    r3,12(r1)

  e_lis     r3,0xFFF48010@ha    ; Load INTC_IACKR,
                                ; which clears
                                ; request to
                                ; processor
  e_lwz     r3,0xFFF48010@l(r3) ; and loads
                                ; address of
                                ; ISR from vector
                                ; table

  ; Code to enable processor recognition of
  ; interrupts and save context required by
  ; EABI
  se_mtctr  r3                  ; Move INTC_IACKR
                                ; contents into
                                ; link register
  ; Set up function addr for the indirect branch
  se_bctr                       ; Indirect branch
                                ; to ISR function
```

```
interrupt_exception_end:
  ; Code to restore context required by EABI and
  ; disable processor recognition of interrupts
  ; Code to restore SRR0 and SRR1

  se_lwz     r3,12(r1)              ; Restore the ctr
  se_mtctr   r3
  se_lwz     r3,8(r1)               ; Restore r3
  se_addi    r1,16                  ; Restore the SP

  se_rfi                            ; Return from the
                                    ; interrupt
```

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:
  se_stw     r0,8(r1)     ; Save interrupt context
                          ; following the EABI

  ; Increment the interrupt counter before enabling
  ;global interrupts
  ; MW55xxVLE_req 708
  se_stw     r4,44(r1)
  e_lis      r4,os_isr_count@ha
  e_lwz      r0,os_isr_count@l(r4)
  se_addi    r0,1
  e_stw      r0,os_isr_count@l(r4)

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

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

```
                          ; Cat 1 blocking ends here

  se_mfctr  r0            ; Save the non GPR regs
  se_stw    r0,20(r1)
  mfspr     r0,xer
  se_stw    r0,24(r1)
  mfcr      r0
  se_stw    r0,28(r1)
  se_mflr   r0
  se_stw    r0,32(r1)

  .ifdef OS_STANDARD_BUILD
  se_mtctr  r3            ; Set up function addr for
                          ; the indirect branch
  .endif ; OS_STANDARD_BUILD

; stw       r3,40(r1)     ; R3 - already done in the
                          ; outer wrapper
; stw       r4,44(r1)
  se_stw    r5,48(r1)
  se_stw    r6,52(r1)
  se_stw    r7,56(r1)
  e_stw     r8,60(r1)
  e_stw     r9,64(r1)
  e_stw     r10,68(r1)
  e_stw     r11,72(r1)
  e_stw     r12,76(r1)

  .ifdef OS_STANDARD_BUILD
  se_bctrl                ; Indirect branch to ISR
                          ; function
  .endif ; OS_STANDARD_BUILD

  e_bl      os_wrapper    ; Call inner wrapper
                          ; and return from
                          ; with R3 holding
                          ; the old IPL

  ; Restore the context
  se_lwz    r0,32(r1)     ; Restore the LR with
                          ; a load inserted
  e_lwz     r12,76(r1)
  se_mtlr   r0

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

  se_lwz    r0,28(r1)     ; Restore the CRF with
                          ; a load inserted
```

```
  se_lwz    r5,48(r1)
  mtcrf     0xff,r0
  se_lwz    r0,24(r1)
  mtspr     xer,r0
  se_lwz    r0,20(r1)
  se_mtctr  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
  e_lis     r4,0xFFF48008@ha
  e_stw     r3,0xFFF48008@l(r4)

  ; Decrement the interrupt counter after enabling
  ; global interrupts
  ; MW55xxVLE_req 709

  e_lis     r4,os_isr_count@ha
  e_lwz     r0,os_isr_count@l(r4)
  se_subi   r0,1
  e_stw     r0,os_isr_count@l(r4)

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

  se_lwz    r0,16(r1)   ; Restore SRR0/1
  mtspr     srr1,r0
  se_lwz    r0,12(r1)
  mtspr     srr0,r0
  se_lwz    r0,8(r1)

  e_addi    r1,r1,80    ; Restore the SP

interrupt_exception_end:
  ; Code to restore context required by EABI and
  ; disable processor recognition of interrupts
  ; Code to restore SRR0 and SRR1

  se_lwz    r3,12(r1)   ; Restore the ctr
  se_mtctr  r3
  se_lwz    r3,8(r1)    ; Restore r3
  se_addi   r1,16       ; Restore the SP

  se_rfi                ; Return from the interrupt
```

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. If however, a z1 core variant is used then there is no distinction between the SW or HW vector tables. The example SW vector handler routine `interrupt_exception_handler`, which is required to be 16-byte aligned (4-byte on a z1), 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 64-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 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 (z1 Core Variants)

The CPU interrupts and exceptions 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,text_vle
  .align      4
  .global     os_CPU_vectors
os_CPU_vectors:
  e_b    Critical_Input               ; IVOR(0)
  e_b    Machine_Check                ; IVOR(1)
  e_b    Data_Storage                 ; IVOR(2)
  e_b    Instruction_Storage          ; IVOR(3)
  e_b    Interrupt_exception_handler  ; IVOR(4)
  e_b    Alignment                    ; IVOR(5)
  e_b    Program                      ; IVOR(6)
  e_b    Floating_Point               ; IVOR(7)
  e_b    System_Call                  ; IVOR(8)
  e_b    Not_Used                     ; IVOR(9)
  e_b    Decrementer                  ; IVOR(10)
  e_b    Fixed_Interval_Timer         ; IVOR(11)
  e_b    Watchdog_Timer               ; IVOR(12)
  e_b    Data_TLB_Error               ; IVOR(13)
  e_b    Instruction_TLB_Error        ; IVOR(14)
  e_b    Debug                        ; IVOR(15)
```

In order to correctly generate the IVOR branch table it is necessary to specify –`DOS_CPUVECTOR_CODE` on the command line when `osgen.s` is assembled. Failure to do this will result in an IVOR table only suitable for use with non z1 core variants and can be distinguished by the use of `.long` as opposed to `e_b` statements.

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 (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 : {
                _start_vec = .;
                os_cpuvec (VLECODE) : {}
                . = _start_vec + 0x800;
                os_intvec (VLECODE) : {}
        } > 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 e200z1 (MPC5514, MPC5516, MPC5517), e200z3 (MPC5534, SPC563M), e200z4 (MPC5643L), e200z6 (MPC5561, MPC5565, MPC5566, MPC5567, MPC5668G), e200z7 (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 (z1 core variants use 0x10800) 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 or z1 core variants use 0x10800 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. This problem has been resolved in this port by the addition of an interrupt nesting counter `osSmallType os_isr_count`, located in 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/Metrowerks 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. This should only be called when interrupts are globally disabled; refer also 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 (z1 core variants use 0x10800) 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 such that the SRR0 and SRR1 registers are saved and restored.

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.
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): 64

Timing

API max usage (bytes): 64

Extended

API max usage (bytes): 64

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. However, because the Metrowerks startup code uses 8-bytes of stack, the stack pointer is kept aligned to an 8-byte boundary. Code generated by the compiler allocates stack in 16-byte multiples. 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 = _stack_addr;
```

3.4 Floating point

Some Freescale PowerPC CPUs (including those based on e200z6 and e200z3 cores) 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>\mw55xxvle\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.

z1 core variants don't include an SPE unit, so it is necessary to either not enable floating-point context saving or define `OS_SPE_NOT_SUPPORTED` when compiling `osfptgt.c` which avoids use of the SPE registers.

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	**MPC5668G**
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	152	152	156	244	244	248
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	224	224	228	316	316	320
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	280	280	284	368	368	372
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	84	84	84	84	84	84
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	34	34	34	34	34	34
Counter	RAM	4	4	4	4	4	4
	ROM	96	96	96	96	96	96
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	82	82	82	82	82	82
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	126	126	126	126	126	126
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	34	34	34	34	34	34
Counter	RAM	4	4	4	4	4	4
	ROM	96	96	96	96	96	96
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	82	82	82	82	82	82
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	138	138	138	138	138	138
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	38	38	38	38	38	38
Counter	RAM	4	4	4	4	4	4
	ROM	100	100	100	100	100	100
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	82	82	82	82	82	82
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	272	400	476	284	412	536
	NS		240	368	444	252	380	504
	KL	2	140	264	340	152	276	400
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	36	36	36	36	36	36
ChainTask	SWL	1, 8	248	376	444	260	388	504
	SWH	1, 9	308	432	500	320	444	556
	NSL	8	248	376	444	260	388	504
	NSH	9	296	420	488	308	432	544
Schedule			216	216	276	216	216	276
GetTaskID			64	64	64	64	64	64
GetTaskState			152	152	152	184	184	184
EnableAllInterrupts			60	60	60	60	60	60
DisableAllInterrupts			116	116	116	116	116	116
ResumeAllInterrupts			88	88	88	88	88	88
SuspendAllInterrupts			156	156	156	156	156	156
ResumeOSInterrupts			116	116	116	116	116	116
SuspendOSInterrupts			164	164	164	164	164	164
GetResource	Task	7	40	40	48	40	40	48
	Combined	6	124	124	124	124	124	124
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	156	156	156	156	156	156
	Combined	6	340	340	340	340	340	340
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	240	240	388
	NS		n/a	n/a	n/a	184	184	332
	NS1i	10	n/a	n/a	n/a	100	n/a	n/a
	KL	2	n/a	n/a	n/a	116	116	264
	KL1i	2, 10	n/a	n/a	n/a	40	n/a	n/a
ClearEvent			n/a	n/a	n/a	96	96	96

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	20	20	20
WaitEvent	<default>		n/a	n/a	n/a	492	492	888
	fp	11	n/a	n/a	n/a	572	572	1048
	1i	10	n/a	n/a	n/a	36	n/a	n/a
GetAlarmBase			96	96	96	96	96	96
GetAlarm			192	192	192	192	192	192
SetRelAlarm			956	956	956	956	956	956
SetAbsAlarm			1036	1036	1036	1036	1036	1036
CancelAlarm			180	180	180	180	180	180
InitCounter			116	116	116	116	116	116
GetCounterValue			140	140	140	140	140	140
GetScheduleTableStatus		34	74	112	112	74	112	112
NextScheduleTable		34	98	288	288	98	288	288
StartScheduleTable		34	134	188	188	134	188	188
StopScheduleTable		34	100	134	134	100	134	134
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		54	54	54	54	54	54
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	40	40	40	40	40	40
Process container	Non-Yielding	33	28	28	28	28	28	28
osek_tick_alarm	<default>		140	140	140	140	140	140
	KL	2	72	72	72	72	72	72
osek_incr_counter			64	64	64	64	64	64
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			376	376	376	376	376	376
ShutdownOS	NoHook	12	56	56	56	56	56	56
	Hook	13	84	84	84	84	84	84
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			76	76	76	76	76	76
StopCOM			36	36	36	36	36	36
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	160	160	160	356	356	356

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	356	356	356	356	356	356
GetMessageResource			104	104	104	104	104	104
ReleaseMessageResource			96	96	96	96	96	96
GetMessageStatus			80	80	80	80	80	80
SendMessage	SW CCCA	1, 14	224	224	224	448	448	448
	SW CCCB	1, 15	416	416	416	448	448	448
	NS CCCA	14	224	224	224	448	448	448
	NS CCCB	15	416	416	416	448	448	448
	KL CCCA	2, 14	124	124	124	348	348	348
	KL CCCB	2, 15	324	324	324	348	348	348
main_dispatch	NoHook	12	316	316	400	316	316	400
	Hook	13	396	396	480	396	396	480
sub_dispatch	B1LF	19	84	84	84	84	84	84
	B1HI	20	240	240	240	240	240	240
	B1HF	21	256	256	256	256	256	256
	B2LI	22	n/a	180	236	n/a	180	236
	B2LF	23	n/a	196	252	n/a	196	252
	B2HI	24	n/a	580	704	n/a	580	704
	B2HF	25	n/a	596	720	n/a	596	720
	E1HI	26	n/a	n/a	n/a	780	780	936
	E1HF	27	n/a	n/a	n/a	796	796	952
	E2HI	28	n/a	n/a	n/a	n/a	n/a	936
	E2HF	29	n/a	n/a	n/a	n/a	n/a	952
ErrorHook support		16	104	104	104	104	104	104
	ServiceID	17	120	120	120	120	120	120
	Parameters	18	152	152	152	152	152	152
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	264	528	616	288	572	724
	NS		232	496	584	256	540	692
	KL	2	132	380	476	156	424	548
ChainTaskset	SWL	1, 8	260	520	600	272	552	704
	SWH	1, 9	348	608	688	360	640	792
	NSL	8	260	520	600	272	552	704
	NSH	9	336	596	676	348	628	780
GetTasksetRef			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	
MergeTaskset			84	84	84	84	84	84
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			88	88	88	88	88	88
TestSubTaskset			100	100	100	100	100	100
TestEquivalentTaskset			96	96	96	96	96	96
TickSchedule	SW	1	392	388	388	388	388	388
	NS		348	340	340	340	340	340
	KL	2	268	264	264	264	264	264
AdvanceSchedule	SW	1	376	368	368	368	368	368
	NS		336	328	328	328	328	328
	KL	2	260	256	256	256	256	256
StartSchedule			188	188	188	188	188	188
StopSchedule			156	156	156	156	156	156
GetScheduleStatus			212	212	212	212	212	212
GetScheduleValue			176	176	176	176	176	176
GetScheduleNext			20	20	20	20	20	20
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			12	12	12	12	12	12
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			12	12	12	12	12	12
TestArrivalpointWritable			68	68	68	68	68	68
GetExecutionTime			8	8	8	8	8	8
GetLargestExecutionTime			16	16	16	16	16	16
ResetLargestExecutionTime			8	8	8	8	8	8
GetStackOffset			80	80	80	80	80	80

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	272	400	476	284	412	536
	NS		240	368	444	252	380	504
	KL	2	140	264	340	152	276	400
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	36	36	36	36	36	36
ChainTask	SWL	1, 8	248	376	444	260	388	504
	SWH	1, 9	308	432	500	320	444	556
	NSL	8	248	376	444	260	388	504
	NSH	9	296	420	488	308	432	544
Schedule			260	260	320	260	260	320
GetTaskID			64	64	64	64	64	64
GetTaskState			152	152	152	184	184	184
EnableAllInterrupts			60	60	60	60	60	60
DisableAllInterrupts			116	116	116	116	116	116
ResumeAllInterrupts			88	88	88	88	88	88
SuspendAllInterrupts			156	156	156	156	156	156
ResumeOSInterrupts			116	116	116	116	116	116
SuspendOSInterrupts			164	164	164	164	164	164
GetResource	Task	7	40	40	48	40	40	48
	Combined	6	124	124	124	124	124	124
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	200	200	200	200	200	200
	Combined	6	428	428	428	428	428	428
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	240	240	388
	NS		n/a	n/a	n/a	184	184	332
	NS1i	10	n/a	n/a	n/a	100	n/a	n/a
	KL	2	n/a	n/a	n/a	116	116	264
	KL1i	2, 10	n/a	n/a	n/a	40	n/a	n/a
ClearEvent			n/a	n/a	n/a	96	96	96
GetEvent			n/a	n/a	n/a	20	20	20
WaitEvent	<default>		n/a	n/a	n/a	628	628	1012
	fp	11	n/a	n/a	n/a	704	704	1172

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	216	n/a	n/a
GetAlarmBase			96	96	96	96	96	96
GetAlarm			192	192	192	192	192	192
SetRelAlarm			956	956	956	956	956	956
SetAbsAlarm			1036	1036	1036	1036	1036	1036
CancelAlarm			180	180	180	180	180	180
InitCounter			116	116	116	116	116	116
GetCounterValue			140	140	140	140	140	140
GetScheduleTableStatus		34	74	112	112	74	112	112
NextScheduleTable		34	98	288	288	98	288	288
StartScheduleTable		34	134	188	188	134	188	188
StopScheduleTable		34	100	134	134	100	134	134
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		54	54	54	54	54	54
GetISRID		4	34	34	34	34	34	34
Process container	Yielding	32	40	40	40	40	40	40
Process container	Non-Yielding	33	28	28	28	28	28	28
osek_tick_alarm	<default>		140	140	140	140	140	140
	KL	2	72	72	72	72	72	72
osek_incr_counter			64	64	64	64	64	64
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			468	468	468	468	468	468
ShutdownOS	NoHook	12	56	56	56	56	56	56
	Hook	13	84	84	84	84	84	84
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			76	76	76	76	76	76
StopCOM			36	36	36	36	36	36
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	160	160	160	356	356	356
	CCCB	15	356	356	356	356	356	356
GetMessageResource			104	104	104	104	104	104
ReleaseMessageResource			96	96	96	96	96	96

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			80	80	80	80	80	80
SendMessage	SW CCCA	1, 14	224	224	224	448	448	448
	SW CCCB	1, 15	416	416	416	448	448	448
	NS CCCA	14	224	224	224	448	448	448
	NS CCCB	15	416	416	416	448	448	448
	KL CCCA	2, 14	124	124	124	348	348	348
	KL CCCB	2, 15	324	324	324	348	348	348
main_dispatch	NoHook	12	400	400	484	400	400	484
	Hook	13	488	488	572	488	488	572
sub_dispatch	B1LF	19	56	56	56	56	56	56
	B1HI	20	216	216	216	216	216	216
	B1HF	21	232	232	232	232	232	232
	B2LI	22	n/a	116	172	n/a	116	172
	B2LF	23	n/a	132	188	n/a	132	188
	B2HI	24	n/a	480	604	n/a	480	604
	B2HF	25	n/a	496	620	n/a	496	620
	E1HI	26	n/a	n/a	n/a	792	792	940
	E1HF	27	n/a	n/a	n/a	808	808	956
	E2HI	28	n/a	n/a	n/a	n/a	n/a	940
	E2HF	29	n/a	n/a	n/a	n/a	n/a	956
ErrorHook support		16	104	104	104	104	104	104
	ServiceID	17	120	120	120	120	120	120
	Parameters	18	152	152	152	152	152	152
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	180	180	180	180	180	180
Timing_termination		4	180	180	180	180	180	180
ActivateTaskset	SW	1	264	528	616	288	572	724
	NS		232	496	584	256	540	692
	KL	2	132	380	476	156	424	548
ChainTaskset	SWL	1, 8	260	520	600	272	552	704
	SWH	1, 9	348	608	688	360	640	792
	NSL	8	260	520	600	272	552	704
	NSH	9	336	596	676	348	628	780
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			84	84	84	84	84	84
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			88	88	88	88	88	88

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestSubTaskset			100	100	100	100	100	100
TestEquivalentTaskset			96	96	96	96	96	96
TickSchedule	SW	1	392	388	388	388	388	388
	NS		348	340	340	340	340	340
	KL	2	268	264	264	264	264	264
AdvanceSchedule	SW	1	376	368	368	368	368	368
	NS		336	328	328	328	328	328
	KL	2	260	256	256	256	256	256
StartSchedule			188	188	188	188	188	188
StopSchedule			156	156	156	156	156	156
GetScheduleStatus			212	212	212	212	212	212
GetScheduleValue			176	176	176	176	176	176
GetScheduleNext			20	20	20	20	20	20
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			12	12	12	12	12	12
GetArrivalpointTasksetRef			12	12	12	12	12	12
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			12	12	12	12	12	12
TestArrivalpointWritable			68	68	68	68	68	68
GetExecutionTime			228	228	228	228	228	228
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			24	24	24	24	24	24
GetStackOffset			80	80	80	80	80	80

Extended

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	512	656	720	524	668	784
	NS		640	784	848	652	796	920
	KL	2	344	460	524	356	472	576
TerminateTask	LExt	3	240	240	240	240	240	240

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	288	288	288	288	288	288
ChainTask	SWL	1, 8	592	736	800	604	748	864
	SWH	1, 9	664	804	868	676	816	932
	NSL	8	736	880	944	748	892	1008
	NSH	9	796	936	1000	808	948	1064
Schedule			516	516	576	516	516	576
GetTaskID			104	104	104	104	104	104
GetTaskState			492	492	492	504	504	504
EnableAllInterrupts			100	100	100	100	100	100
DisableAllInterrupts			156	156	156	156	156	156
ResumeAllInterrupts			192	192	192	192	192	192
SuspendAllInterrupts			196	196	196	196	196	196
ResumeOSInterrupts			192	192	192	192	192	192
SuspendOSInterrupts			204	204	204	204	204	204
GetResource	Task	7	764	764	676	764	764	676
	Combined	6	748	748	748	748	748	748
	CLEx	3	620	620	620	620	620	620
ReleaseResource	Task	7	692	692	692	692	692	692
	Combined	6	920	920	920	920	920	920
	CLEx	3	636	636	636	636	636	636
SetEvent	SW	1	n/a	n/a	n/a	592	592	748
	NS		n/a	n/a	n/a	716	716	876
	NS1i	10	n/a	n/a	n/a	492	n/a	n/a
	KL	2	n/a	n/a	n/a	424	424	576
	KL1i	2, 10	n/a	n/a	n/a	356	n/a	n/a
ClearEvent			n/a	n/a	n/a	272	272	272
GetEvent			n/a	n/a	n/a	328	328	328
WaitEvent	<default>		n/a	n/a	n/a	828	828	1180
	fp	11	n/a	n/a	n/a	912	912	1356
	1i	10	n/a	n/a	n/a	444	n/a	n/a
GetAlarmBase			388	388	388	388	388	388
GetAlarm			376	376	376	376	376	376
SetRelAlarm			1216	1216	1216	1216	1216	1216
SetAbsAlarm			1256	1256	1256	1256	1256	1256
CancelAlarm			348	348	348	348	348	348
InitCounter			464	464	464	464	464	464
GetCounterValue			432	432	432	432	432	432

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	98	136	136	98	136	136
NextScheduleTable		34	122	312	312	122	312	312
StartScheduleTable		34	158	212	212	158	212	212
StopScheduleTable		34	124	158	158	124	158	158
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		54	54	54	54	54	54
GetISRID		4	58	58	58	58	58	58
Process container	Yielding	32	40	40	40	40	40	40
Process container	Non-Yielding	33	28	28	28	28	28	28
osek_tick_alarm	<default>		268	268	268	268	268	268
	KL	2	72	72	72	72	72	72
osek_incr_counter			64	64	64	64	64	64
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			508	508	508	508	508	508
ShutdownOS	NoHook	12	76	76	76	76	76	76
	Hook	13	104	104	104	104	104	104
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			116	116	116	116	116	116
StopCOM			100	100	100	100	100	100
ReadFlag			64	64	64	64	64	64
ResetFlag			76	76	76	76	76	76
ReceiveMessage	CCCA	14	356	356	356	544	544	544
	CCCB	15	544	544	544	544	544	544
GetMessageResource			216	216	216	216	216	216
ReleaseMessageResource			224	224	224	224	224	224
GetMessageStatus			212	212	212	212	212	212
SendMessage	SW CCCA	1, 14	448	448	448	672	672	672
	SW CCCB	1, 15	640	640	640	672	672	672
	NS CCCA	14	448	448	448	672	672	672
	NS CCCB	15	640	640	640	672	672	672
	KL CCCA	2, 14	300	300	300	512	512	512
	KL CCCB	2, 15	488	488	488	512	512	512
main_dispatch	NoHook	12	400	400	484	400	400	484

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	488	488	572	488	488	572
sub_dispatch	B1LF	19	56	56	56	56	56	56
	B1HI	20	216	216	216	216	216	216
	B1HF	21	232	232	232	232	232	232
	B2LI	22	n/a	116	172	n/a	116	172
	B2LF	23	n/a	132	188	n/a	132	188
	B2HI	24	n/a	480	604	n/a	480	604
	B2HF	25	n/a	496	620	n/a	496	620
	E1HI	26	n/a	n/a	n/a	792	792	940
	E1HF	27	n/a	n/a	n/a	808	808	956
	E2HI	28	n/a	n/a	n/a	n/a	n/a	940
	E2HF	29	n/a	n/a	n/a	n/a	n/a	956
ErrorHook support		16	256	256	256	256	256	256
	ServiceID	17	272	272	272	272	272	272
	Parameters	18	304	304	304	304	304	304
validity_checks		3	44	44	44	44	44	44
Timing_dispatch		4	180	180	180	180	180	180
Timing_termination		4	180	180	180	180	180	180
ActivateTaskset	SW	1	664	780	856	688	824	948
	NS		788	924	1000	812	968	1088
	KL	2	480	596	672	508	640	764
ChainTaskset	SWL	1, 8	776	880	956	788	912	1036
	SWH	1, 9	864	988	1068	876	1024	1144
	NSL	8	916	1040	1116	928	1072	1192
	NSH	9	992	1120	1196	1008	1152	1272
GetTasksetRef			288	288	288	288	288	288
MergeTaskset			600	600	600	600	600	600
AssignTaskset			384	384	384	384	384	384
RemoveTaskset			604	604	604	604	604	604
TestSubTaskset			624	624	624	624	624	624
TestEquivalentTaskset			620	620	620	620	620	620
TickSchedule	SW	1	628	600	600	600	600	600
	NS		756	748	748	748	748	748
	KL	2	484	460	460	460	460	460
AdvanceSchedule	SW	1	660	628	628	628	628	628
	NS		788	776	776	776	776	776
	KL	2	520	488	488	488	488	488

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			516	516	516	516	516	516
StopSchedule			412	412	412	412	412	412
GetScheduleStatus			476	476	476	476	476	476
GetScheduleValue			396	396	396	396	396	396
GetScheduleNext			176	176	176	176	176	176
SetScheduleNext			344	344	344	344	344	344
GetArrivalpointDelay			264	264	264	264	264	264
SetArrivalpointDelay			292	292	292	292	292	292
GetArrivalpointTasksetRef			260	260	260	260	260	260
GetArrivalpointNext			264	264	264	264	264	264
SetArrivalpointNext			372	372	372	372	372	372
TestArrivalpointWritable			316	316	316	316	316	316
GetExecutionTime			332	332	332	332	332	332
GetLargestExecutionTime			232	232	232	232	232	232
ResetLargestExecutionTime			220	220	220	220	220	220
GetStackOffset			80	80	80	80	80	80

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	340	498	573	353	464	577
	NS	296	447	535	300	430	579
	KL	217	352	442	213	334	459
TerminateTask	LExt	0	0	0	0	0	0
	H	657	642	636	656	646	655
ChainTask	SWL	949	1165	1375	1262	1364	1584
	SWH	1216	1417	1635	1510	1599	1833
	NSL	947	1163	1368	1249	1355	1575
	NSH	1209	1407	1625	1492	1592	1811
Schedule	SW	294	310	363	309	296	353
GetTaskID		128	139	139	139	133	135
GetTaskState		232	234	237	275	267	274
EnableAllInterrupts		109	109	121	109	109	109
DisableAllInterrupts		205	222	231	214	203	206
ResumeAllInterrupts		149	133	129	133	152	149
SuspendAllInterrupts		250	252	265	236	236	250
ResumeOSInterrupts		133	133	129	133	136	133
SuspendOSInterrupts		250	236	249	252	236	250
GetResource	Task	135	140	136	135	140	139
	Combined	217	208	213	205	211	217
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	231	224	228	232	228	224
	Combined	375	368	386	376	386	368
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	325	321	314
	NS	n/a	n/a	n/a	248	276	268
	KL	n/a	n/a	n/a	181	182	179
ClearEvent		n/a	n/a	n/a	175	157	157
GetEvent		n/a	n/a	n/a	78	80	80
WaitEvent	<default>	n/a	n/a	n/a	2208	2192	2419
	fp	n/a	n/a	n/a	2261	2234	2503
GetAlarmBase		210	190	204	204	209	209
GetAlarm		277	261	271	271	270	270

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		517	502	500	513	516	516
SetAbsAlarm		488	467	478	491	497	497
CancelAlarm		244	241	228	228	231	231
InitCounter		200	214	214	214	203	202
GetCounterValue		208	235	235	235	216	213
osek_tick_alarm	<default>	227	217	215	215	232	232
	KL	174	161	171	171	160	160
osek_incr_counter		46	43	43	43	44	44
GetActiveApplicationMode		23	28	28	28	28	27
StartOS		4575	4421	4549	4421	4533	4437
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	151	143	151	143	151	144
InitCOM		57	73	65	65	57	57
CloseCOM		49	65	65	73	57	66
StartCOM		199	191	192	554	561	577
StopCOM		75	81	81	81	75	75
ReadFlag		n/a	n/a	n/a	54	52	51
ResetFlag		n/a	n/a	n/a	43	43	43
ReceiveMessage		233	233	234	940	924	944
GetMessageResource		n/a	n/a	n/a	330	317	320
ReleaseMessageResource		n/a	n/a	n/a	406	417	392
GetMessageStatus		n/a	n/a	n/a	170	174	174
SendMessage	SW	643	797	867	1337	1457	1578
	NS	598	759	843	1293	1407	1564
	KL	408	560	649	1110	1239	1372
ActivateTaskset	SW	296	2378	2735	289	2393	3056
	NS	242	2334	2458	251	2320	2503
	KL	150	2493	2631	169	2257	2433
	SW2	295	2378	2734	288	2393	3056
	NS2	242	2333	2458	251	2320	2503
	KL2	150	2493	2631	169	2257	2433
ChainTaskset	SWL	894	3050	3292	1197	3251	3532
	SWH	1194	3588	3580	1480	3795	3817
	NSL	918	3297	3287	1203	3507	4035
	NSH	1188	3340	3808	1455	3542	4317
GetTasksetRef		85	88	93	93	80	85

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		169	158	160	160	166	168
AssignTaskset		75	84	83	83	67	67
RemoveTaskset		169	144	154	154	144	154
TestSubTaskset		168	170	158	158	178	166
TestEquivalentTaskset		161	158	153	153	166	161
TickSchedule	SW	501	2972	3122	643	2775	2926
	NS	438	2926	3072	596	2717	2872
	KL	350	2827	2949	497	2602	2781
	SW2	501	2972	3122	642	2756	2920
	NS2	437	2926	3072	596	2698	2866
	KL2	350	2827	2949	497	2583	2775
AdvanceSchedule	SW	525	3034	3185	712	2826	2989
	NS	401	2984	3124	650	2777	2940
	KL	312	2905	3018	545	2678	2841
	SW2	525	3034	3184	711	2806	2982
	NS2	401	2984	3123	650	2758	2934
	KL2	312	2905	3018	545	2659	2835
StartSchedule		314	312	312	313	306	306
StopSchedule		275	273	261	273	263	275
GetScheduleStatus		281	284	271	284	271	284
GetScheduleValue		286	301	299	301	307	309
GetScheduleNext		94	86	86	86	86	86
SetScheduleNext		75	70	75	70	85	80
GetArrivalpointDelay		84	76	81	81	81	81
SetArrivalpointDelay		61	61	61	61	77	71
GetArrivalpointTasksetRef		81	73	67	67	67	67
GetArrivalpointNext		67	67	67	62	75	70
SetArrivalpointNext		61	71	77	71	77	77
TestArrivalpointWritable		97	97	99	99	91	99
GetExecutionTime		57	49	50	57	65	74
GetLargestExecutionTime		89	81	83	75	85	79
ResetLargestExecutionTime		49	49	57	49	63	63
GetStackOffset		151	161	153	153	154	154

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	346	469	565	335	466	579
	NS	309	432	533	301	419	571
	KL	201	350	444	208	340	467
TerminateTask	LExt	0	0	0	0	0	0
	H	1463	1436	1462	1421	1450	1459
ChainTask	SWL	1879	2079	2273	2146	2281	2515
	SWH	2091	2280	2504	2378	2508	2729
	NSL	1879	2077	2268	2158	2289	2520
	NSH	2091	2286	2512	2379	2494	2717
Schedule	SW	302	296	352	302	296	338
GetTaskID		127	133	136	127	136	128
GetTaskState		244	223	229	275	268	266
EnableAllInterrupts		125	109	109	109	125	125
DisableAllInterrupts		210	213	210	226	213	222
ResumeAllInterrupts		136	133	136	136	133	133
SuspendAllInterrupts		236	236	236	260	250	244
ResumeOSInterrupts		152	133	136	136	150	149
SuspendOSInterrupts		236	252	252	244	250	244
GetResource	Task	144	136	139	141	137	147
	Combined	200	205	200	203	206	205
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	232	236	237	218	231	231
	Combined	389	383	391	375	399	399
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	328	312	309
	NS	n/a	n/a	n/a	268	260	262
	KL	n/a	n/a	n/a	174	179	185
ClearEvent		n/a	n/a	n/a	163	169	161
GetEvent		n/a	n/a	n/a	86	78	78
WaitEvent	<default>	n/a	n/a	n/a	3040	3055	3316
	fp	n/a	n/a	n/a	2887	2917	3389
GetAlarmBase		210	204	204	202	198	210
GetAlarm		280	279	279	269	277	277

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		497	512	512	503	501	502
SetAbsAlarm		493	499	499	475	472	473
CancelAlarm		231	236	236	236	257	244
InitCounter		200	203	202	200	202	200
GetCounterValue		208	222	219	214	213	208
osek_tick_alarm	<default>	220	228	228	240	217	227
	KL	171	176	176	165	174	173
osek_incr_counter		44	36	36	36	46	46
GetActiveApplicationMode		27	19	19	19	22	22
StartOS		12071	12049	11872	12071	11884	13918
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	151	141	145	149	151	151
InitCOM		65	57	65	65	49	49
CloseCOM		57	57	57	57	49	49
StartCOM		207	199	199	553	553	560
StopCOM		75	75	75	75	78	78
ReadFlag		n/a	n/a	n/a	51	47	47
ResetFlag		n/a	n/a	n/a	37	46	46
ReceiveMessage		241	242	241	962	941	1022
GetMessageResource		n/a	n/a	n/a	313	328	326
ReleaseMessageResource		n/a	n/a	n/a	424	430	416
GetMessageStatus		n/a	n/a	n/a	168	162	154
SendMessage	SW	671	783	879	1340	1474	1654
	NS	630	737	838	1302	1438	1657
	KL	412	550	644	1110	1254	1441
ActivateTaskset	SW	276	2604	2734	301	2357	3048
	NS	247	2575	2466	263	2581	2503
	KL	157	2238	2623	145	2243	2433
	SW2	276	2604	2734	301	2357	3048
	NS2	247	2575	2466	263	2581	2503
	KL2	157	2238	2623	145	2243	2433
ChainTaskset	SWL	1847	4209	4190	2112	4422	4465
	SWH	2065	4211	4449	2347	4680	4711
	NSL	1851	3970	4185	2115	4447	4968
	NSH	2077	4444	4676	2335	4428	5210
GetTasksetRef		85	91	91	86	85	85

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
MergeTaskset		178	161	152	150	170	160
AssignTaskset		63	67	67	67	70	75
RemoveTaskset		149	161	162	153	165	170
TestSubTaskset		178	168	166	178	178	166
TestEquivalentTaskset		177	169	153	158	161	145
TickSchedule	SW	493	2737	3110	639	2757	2938
	NS	437	2679	3056	597	2688	2869
	KL	350	2572	2973	481	2597	2778
	SW2	493	2737	3110	639	2742	2932
	NS2	437	2679	3056	597	2673	2863
	KL2	350	2572	2973	481	2582	2772
AdvanceSchedule	SW	563	2794	3179	701	2798	2993
	NS	391	2733	3118	637	2765	2945
	KL	301	2628	3013	538	2686	2846
	SW2	563	2794	3179	701	2783	2987
	NS2	391	2733	3118	637	2750	2939
	KL2	301	2628	3013	538	2671	2840
StartSchedule		316	312	312	322	299	299
StopSchedule		269	255	267	261	273	273
GetScheduleStatus		273	271	284	273	290	290
GetScheduleValue		302	323	325	302	294	294
GetScheduleNext		86	86	86	86	94	94
SetScheduleNext		85	91	86	91	75	75
GetArrivalpointDelay		81	76	81	81	84	84
SetArrivalpointDelay		77	77	83	83	61	61
GetArrivalpointTasksetRef		67	73	67	67	81	81
GetArrivalpointNext		75	83	83	83	67	67
SetArrivalpointNext		77	61	61	61	61	61
TestArrivalpointWritable		91	105	99	99	97	97
GetExecutionTime		440	431	433	432	440	441
GetLargestExecutionTime		109	116	116	116	118	118
ResetLargestExecutionTime		100	86	94	86	78	78
GetStackOffset		142	167	167	168	151	160

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	1246	1414	1504	1198	1401	1520
	NS	1375	1543	1628	1318	1544	1691
	KL	1035	1157	1225	986	1169	1281
TerminateTask	LExt	1720	1716	1735	1740	1712	1704
	H	1909	1902	1908	1928	1865	1923
ChainTask	SWL	3108	3296	3470	3272	3453	3770
	SWH	3317	3508	3700	3488	3677	3960
	NSL	3267	3481	3622	3412	3603	3902
	NSH	3447	3660	3855	3626	3801	4103
Schedule	SW	566	553	601	558	552	623
GetTaskID		147	153	157	147	165	162
GetTaskState		1254	1272	1264	1199	1254	1271
EnableAllInterrupts		135	134	135	151	143	143
DisableAllInterrupts		241	242	233	241	242	250
ResumeAllInterrupts		210	227	226	210	206	206
SuspendAllInterrupts		262	282	262	262	265	273
ResumeOSInterrupts		210	211	210	227	206	206
SuspendOSInterrupts		278	282	262	262	281	289
GetResource	Task	2403	2483	1134	2515	2597	1285
	Combined	1076	1070	1079	1181	1212	1220
	CLEx	1140	1132	1136	1242	1267	1296
ReleaseResource	Task	1066	1064	1081	1177	1189	1216
	Combined	1105	1082	1082	1216	1228	1234
	CLEx	1070	1064	1028	1161	1160	1186
SetEvent	SW	n/a	n/a	n/a	1300	1338	1366
	NS	n/a	n/a	n/a	1335	1358	1372
	KL	n/a	n/a	n/a	1076	1159	1163
ClearEvent		n/a	n/a	n/a	337	320	342
GetEvent		n/a	n/a	n/a	970	1034	1032
WaitEvent	<default>	n/a	n/a	n/a	3615	3518	3567
	fp	n/a	n/a	n/a	3431	3339	3601
GetAlarmBase		845	854	893	819	835	893
GetAlarm		874	885	922	848	864	914

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		1210	1208	1251	1177	1167	1245
SetAbsAlarm		1137	1130	1177	1103	1095	1164
CancelAlarm		837	846	885	811	812	877
InitCounter		1435	1488	1535	1406	1454	1588
GetCounterValue		813	826	800	778	808	815
osek_tick_alarm	<default>	434	415	421	421	422	421
	KL	164	184	172	172	160	159
osek_incr_counter		36	46	46	46	43	43
GetActiveApplicationMode		19	22	22	22	27	27
StartOS		14374	14374	12257	14543	12437	12268
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	159	161	161	161	153	161
InitCOM		57	50	49	49	65	65
CloseCOM		65	57	49	57	65	65
StartCOM		236	236	247	618	608	547
StopCOM		108	108	108	108	114	120
ReadFlag		n/a	n/a	n/a	125	123	113
ResetFlag		n/a	n/a	n/a	111	117	117
ReceiveMessage		714	706	728	1474	1413	1452
GetMessageResource		n/a	n/a	n/a	1688	1723	1730
ReleaseMessageResource		n/a	n/a	n/a	1610	1677	1672
GetMessageStatus		n/a	n/a	n/a	498	520	522
SendMessage	SW	2055	2214	2268	2740	2882	3002
	NS	2167	2340	2389	2852	3023	3171
	KL	1633	1750	1808	2343	2468	2561
ActivateTaskset	SW	2714	5047	5418	2695	5283	4990
	NS	2797	5177	5047	2803	5175	5352
	KL	2734	4582	4941	2462	4817	4762
	SW2	2714	5047	5418	2695	5283	4990
	NS2	2797	5177	5047	2803	5175	5352
	KL2	2734	4582	4941	2462	4817	4762
ChainTaskset	SWL	4878	7222	6960	4833	7131	7755
	SWH	4852	7464	7710	5067	7363	7746
	NSL	5014	7387	7615	5233	7291	8135
	NSH	4958	7590	7862	5462	7744	7888
GetTasksetRef		962	963	955	897	969	953

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		569	566	566	580	550	558
AssignTaskset		301	305	305	310	297	301
RemoveTaskset		565	582	582	547	566	563
TestSubTaskset		702	697	697	690	694	697
TestEquivalentTaskset		700	685	685	678	690	682
TickSchedule	SW	783	5305	5648	3173	5545	5508
	NS	920	5383	5768	3277	5691	5594
	KL	546	5056	5409	2928	5314	5275
	SW2	783	5305	5648	3173	5505	5477
	NS2	920	5383	5768	3277	5651	5563
	KL2	546	5056	5409	2928	5274	5244
AdvanceSchedule	SW	870	5378	5733	3254	5660	5598
	NS	998	5492	5858	3379	5771	5707
	KL	542	5148	5513	3034	5409	5362
	SW2	870	5378	5733	3254	5620	5567
	NS2	998	5492	5858	3379	5731	5676
	KL2	542	5148	5513	3034	5369	5331
StartSchedule		661	647	635	655	647	655
StopSchedule		477	483	495	485	471	469
GetScheduleStatus		496	512	510	496	498	504
GetScheduleValue		538	514	510	522	517	522
GetScheduleNext		180	181	185	180	177	173
SetScheduleNext		356	376	360	350	369	392
GetArrivalpointDelay		269	277	276	267	255	269
SetArrivalpointDelay		306	290	324	309	327	306
GetArrivalpointTasksetRef		212	220	219	224	237	212
GetArrivalpointNext		239	223	214	229	225	239
SetArrivalpointNext		362	362	404	395	383	362
TestArrivalpointWritable		261	253	246	245	260	261
GetExecutionTime		588	588	603	610	579	608
GetLargestExecutionTime		918	915	894	834	918	907
ResetLargestExecutionTime		870	862	851	809	870	866
GetStackOffset		168	160	151	142	168	156

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	160	160	160	160	160	160
	Cat 2	302	632	615	627	635	628

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	169	169	169	169	169	169
	Cat 2	1035	1376	1376	1358	1355	1359

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	169	169	169	169	169	169
	Cat 2	1034	1383	1374	1383	1371	1358

4.3.4 Task Switching Times

Task switching time is the time between the last instruction of the previous task and the first instruction of the next task. The switching time differs, depending on the switching contexts (e.g. an `ActivateTask()` versus a `ChainTask()`).

RTA-OSEK sub-task types also affect the switching time. The tables in this section show the switching times (in CPU cycles) for all system classes for basic, lightweight tasks and for basic and extended heavyweight tasks.

Figures 1 to 8 show the RTA-OSEK switching contexts measured.

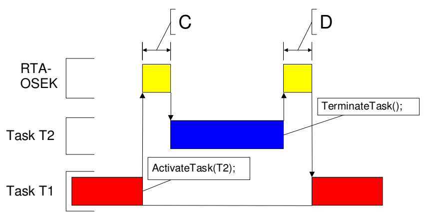

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

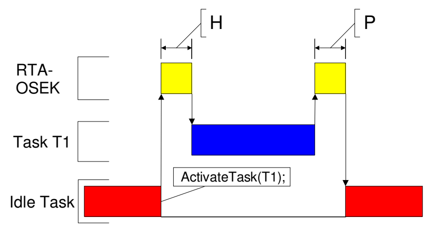

Figure 3: Task Activation from Idle Task

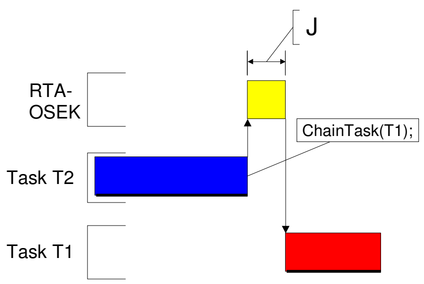

Figure 2: Task Chaining

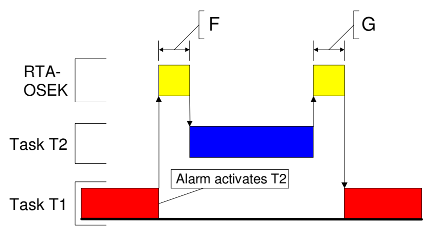

Figure 4: Task Activation from an Alarm

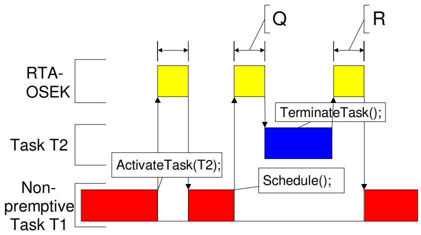

Figure 5: Non-Premptive Task Calls Schedule()

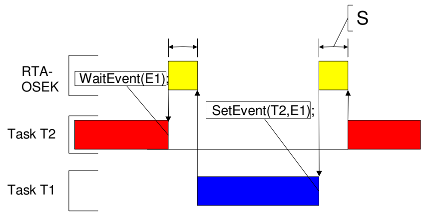

Figure 7: Waiting Task Activated by SetEvent()

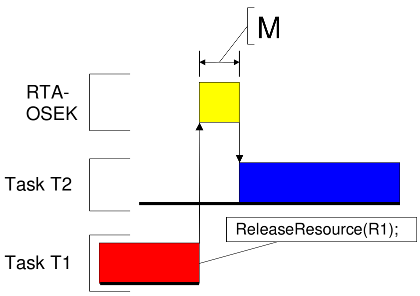

Figure 6: Blocked Task Activated by ReleaseResource()

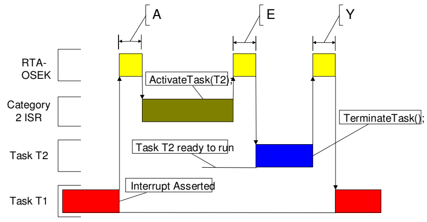

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	338	482	544	343	485	559
Figure 1: D	Heavy, Basic/Extended	657	763	816	818	797	883
ChainTask	Light, Basic	652	862	1073	680	870	1121
Figure 2: J	Heavy, Basic/Extended	1781	2074	2346	1943	2099	2453
Pre-emption	Light, Basic	545	765	983	552	748	1017
Figure 1: C	Heavy, Basic/Extended	872	1098	1315	1164	1276	1508
From idle task	Light, Basic	535	757	975	544	740	1009
Figure 3: H	Heavy, Basic/Extended	862	1090	1307	1156	1268	1500
Triggered by alarm	Light, Basic	825	1021	1253	806	1006	1291
Figure 4: F	Heavy, Basic/Extended	1136	1370	1569	1426	1558	1774
Schedule	Light, Basic	462	536	730	470	534	731
Figure 5: Q	Heavy, Basic/Extended	789	869	1062	1082	1069	1240
Release resource	Light, Basic	478	532	691	466	546	673
Figure 6: M	Heavy, Basic/Extended	805	865	1023	1078	1081	1182
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	2284	2331	2761
From category 2 ISR	Light, Basic	528	942	1094	879	936	1084
Figure 8: E	Heavy, Basic/Extended	855	1275	1426	1491	1471	1593

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	1191	1246	1337	1162	1259	1353
Figure 1: D	Heavy, Basic/Extended	1463	1495	1571	1530	1552	1625
ChainTask	Light, Basic	1588	1771	1962	1603	1775	2050
Figure 2: J	Heavy, Basic/Extended	3448	3660	3949	3570	3762	4089
Pre-emption	Light, Basic	1109	1290	1524	1129	1298	1543
Figure 1: C	Heavy, Basic/Extended	1415	1628	1865	1702	1818	2035
From idle task	Light, Basic	1101	1282	1516	1121	1288	1533
Figure 3: H	Heavy, Basic/Extended	1407	1620	1857	1694	1808	2025
Triggered by alarm	Light, Basic	1379	1552	1770	1411	1568	1807
Figure 4: F	Heavy, Basic/Extended	1661	1882	2135	1968	2072	2315
Schedule	Light, Basic	1035	1072	1274	1045	1075	1249
Figure 5: Q	Heavy, Basic/Extended	1341	1410	1615	1618	1617	1759
Release resource	Light, Basic	1041	1072	1231	1061	1081	1213
Figure 6: M	Heavy, Basic/Extended	1347	1410	1572	1634	1623	1723
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	2896	2880	3375
From category 2 ISR	Light, Basic	1928	2286	2436	2267	2292	2421
Figure 8: E	Heavy, Basic/Extended	2214	2612	2757	2820	2822	2919

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	1688	1790	1887	1727	1780	1843
Figure 1: D	Heavy, Basic/Extended	1908	1980	2039	2047	1968	2090
ChainTask	Light, Basic	2796	3004	3193	2740	2986	3281
Figure 2: J	Heavy, Basic/Extended	5094	5377	5655	5176	5355	5754
Pre-emption	Light, Basic	1967	2162	2358	1897	2154	2439
Figure 1: C	Heavy, Basic/Extended	2294	2484	2665	2459	2659	2950
From idle task	Light, Basic	1959	2154	2350	1889	2146	2431
Figure 3: H	Heavy, Basic/Extended	2286	2476	2657	2451	2651	2942
Triggered by alarm	Light, Basic	2443	2624	2810	2373	2623	2891
Figure 4: F	Heavy, Basic/Extended	2746	2938	3141	2911	3120	3426
Schedule	Light, Basic	1241	1271	1434	1220	1251	1477
Figure 5: Q	Heavy, Basic/Extended	1568	1593	1741	1782	1763	2006
Release resource	Light, Basic	1771	1772	1899	1857	1921	2069
Figure 6: M	Heavy, Basic/Extended	2098	2094	2206	2419	2433	2598
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	3945	3907	4227
From category 2 ISR	Light, Basic	2087	2443	2576	2417	2432	2621
Figure 8: E	Heavy, Basic/Extended	2390	2741	2871	2955	2924	3138

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	160	160	160	160	160	160
BCC1 lightweight, floating-point	176	176	176	176	176	176
BCC1 heavyweight, integer	304	304	304	304	304	304
BCC1 heavyweight, floating-point	304	304	304	304	304	304
BCC2 lightweight, integer	n/a	192	192	n/a	192	192
BCC2 lightweight, floating-point	n/a	192	192	n/a	192	192
BCC2 heavyweight, integer	n/a	304	304	n/a	304	304
BCC2 heavyweight, floating-point	n/a	304	304	n/a	304	304
ECC1 heavyweight, integer	n/a	n/a	n/a	352	352	304
ECC1 heavyweight, floating-point	n/a	n/a	n/a	352	352	304
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	304
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	304
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	160	160	160	160	160	160
BCC1 lightweight, floating-point	176	176	176	176	176	176
BCC1 heavyweight, integer	304	304	304	304	304	304
BCC1 heavyweight, floating-point	304	304	304	304	304	304
BCC2 lightweight, integer	n/a	192	192	n/a	192	192
BCC2 lightweight, floating-point	n/a	192	192	n/a	192	192
BCC2 heavyweight, integer	n/a	304	304	n/a	304	304
BCC2 heavyweight, floating-point	n/a	304	304	n/a	304	304
ECC1 heavyweight, integer	n/a	n/a	n/a	304	304	304
ECC1 heavyweight, floating-point	n/a	n/a	n/a	304	304	304
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	304
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	304

Timing

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Pre- and Post-Task hooks not used						
Task type						
BCC1 lightweight, integer	208	208	208	208	208	208
BCC1 lightweight, floating-point	224	224	224	224	224	224
BCC1 heavyweight, integer	352	352	352	352	352	352
BCC1 heavyweight, floating-point	352	352	352	352	352	352
BCC2 lightweight, integer	n/a	224	224	n/a	224	224
BCC2 lightweight, floating-point	n/a	224	224	n/a	224	224
BCC2 heavyweight, integer	n/a	352	352	n/a	352	352
BCC2 heavyweight, floating-point	n/a	352	352	n/a	352	352
ECC1 heavyweight, integer	n/a	n/a	n/a	400	400	352
ECC1 heavyweight, floating-point	n/a	n/a	n/a	400	400	352
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	352
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	352
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	208	208	208	208	208	208
BCC1 lightweight, floating-point	224	224	224	224	224	224
BCC1 heavyweight, integer	352	352	352	352	352	352
BCC1 heavyweight, floating-point	352	352	352	352	352	352
BCC2 lightweight, integer	n/a	224	224	n/a	224	224
BCC2 lightweight, floating-point	n/a	224	224	n/a	224	224
BCC2 heavyweight, integer	n/a	352	352	n/a	352	352
BCC2 heavyweight, floating-point	n/a	352	352	n/a	352	352
ECC1 heavyweight, integer	n/a	n/a	n/a	352	352	352
ECC1 heavyweight, floating-point	n/a	n/a	n/a	352	352	352
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	352
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	352

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	208	208	208	208	208	208
BCC1 lightweight, floating-point	224	224	224	224	224	224
BCC1 heavyweight, integer	352	352	352	352	352	352
BCC1 heavyweight, floating-point	352	352	352	352	352	352
BCC2 lightweight, integer	n/a	224	224	n/a	224	224
BCC2 lightweight, floating-point	n/a	224	224	n/a	224	224
BCC2 heavyweight, integer	n/a	352	352	n/a	352	352
BCC2 heavyweight, floating-point	n/a	352	352	n/a	352	352
ECC1 heavyweight, integer	n/a	n/a	n/a	400	400	352
ECC1 heavyweight, floating-point	n/a	n/a	n/a	400	400	352
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	352
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	352
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	208	208	208	208	208	208
BCC1 lightweight, floating-point	224	224	224	224	224	224
BCC1 heavyweight, integer	352	352	352	352	352	352
BCC1 heavyweight, floating-point	352	352	352	352	352	352
BCC2 lightweight, integer	n/a	224	224	n/a	224	224
BCC2 lightweight, floating-point	n/a	224	224	n/a	224	224
BCC2 heavyweight, integer	n/a	352	352	n/a	352	352
BCC2 heavyweight, floating-point	n/a	352	352	n/a	352	352
ECC1 heavyweight, integer	n/a	n/a	n/a	352	352	352
ECC1 heavyweight, floating-point	n/a	n/a	n/a	352	352	352
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	352
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	352

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for the MPC55xxVLE/Metrowerks 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 5.0.0

6.1 Variants

The supported target cores in the RTA-OSEK GUI are the e200z1 (MPC5514, MPC5516, MPC5517), e200z3 (MPC5534, SPC563M), e200z4 (MPC5643L), e200z6 (MPC5561, MPC5565, MPC5566, MPC5567, MPC5668G), e200z7 (MPC5674F) and the MPC55xx VLE Generic.

6.2 OS_FIFO_LOAD()

A macro `OS_LIFO_LOAD()` exists that 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.

6.3 CPU Vector Generation Setting

The use of `–DOS_CPUVECTOR_CODE` on the command line when assembling `osgen.s`, is required when using z1 core variants because they require a CPU vector table consisting of branch instructions, as opposed to the more traditional `.long` statements. The default behavior will generate `.long` statements suitable for use with non z1 core variants.

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.