RTA-OSEK

Binding Manual: MPC55xx/GreenHills

Contact Details

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

Copyright Notice

Version: M00081-005

Disclaimer

Trademarks

Safety Notice

This ETAS product fulfills standard quality management requirements. If requirements of specific safety standards (e.g. IEC 61508, ISO 26262) need to be fulfilled, these requirements must be explicitly defined and ordered by the customer. Before use of the product, customer must verify the compliance with specific safety standards.

Contents

1 About this Guide

This guide provides target-specific information for the MPC55xx/GreenHills 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 MPC55xx/GreenHills port supports the single flat memory model supported by the Green Hills Software, Inc. toolchain. This toolchain supports the Embedded Application Binary Interface, EABI.

2.1 Compiler

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

Vendor	Green Hills Software, Inc.
Compiler	Multi C/C++ Compiler for PowerPC
Version	2013.1.4p

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

Option	**Description**
`-cpu=%CPU_TYPE%`	Selects the correct target for code generation etc.
`-noSPE`	Prevent compiler from using vector instructions in function calls

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**
`-cpu=%CPU_TYPE%`	Selects the correct target for code generation etc.
`-no_debug`	Turn debug mode off.
`-noSPE`	Prevent compiler from using vector instructions in function calls
`-sda=0`	Enable SDA, but place only named variables in SDA

To support the use of multiple CPU configurations the environment variable CPU_TYPE must be set up to match the desired CPU target (e.g. ppc5554).

2.2 Assembler

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

Vendor	Green Hills Software, Inc.
Assembler	Green Hills Assembler for the PowerPC
Version	2013.1.4p

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

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

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

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

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

To support the use of multiple CPU configurations the environment variable CPU_TYPE must be set up to match the desired CPU target (e.g. ppc5554).

The `-noSPE` flag is mandatory in the *compiler* to prevent the compiler from emitting code that uses the full 64-bit GPRs. However, application code is allowed to use 64-bit GPRs – see the section on "Floating Point Wrappers" for more information. If vector instructions are used `-noSPE` must **not** be passed to the assembler for the floating-point wrappers, nor the application code containing the vector instructions. It remains mandatory for the compiler. If the assembler is invoked through the compiler driver then SPE support can be turned on for the assembler by putting `-asm="-SPE"` on the compiler command line.

Important: If `-noSPE` is passed to the assembler when compiling vector instructions, a syntax error is thrown.

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	Vector table (if generated by RTA-OSEK). Should be aligned on a 4 or 64KByte boundary.
`os_text`	ROM	RTA-OSEK code section.
`os_pir`	RAM	RTA-OSEK initialized data. Initialized by StartOS(). Can be located in 'far' RAM.
`os_pir2`	RAM	RTA-OSEK initialized data. Must be initialized during C-startup. Can be located in 'far' RAM.
`os_pnir`	RAM	RTA-OSEK near initialized data. Initialized by StartOS(). Must be placed in the compiler SDA (near addressing).
`os_pnir2`	RAM	RTA-OSEK near initialized data. Must be initialized during C-startup. Must be placed in the compiler SDA (near addressing).
`os_cntr`	RAM	RTA-OSEK near uninitialized data. Must be zeroed during C-startup. Must be placed in the compiler SDA (near addressing).
`os_cntri`	RAM	RTA-OSEK near initialized (with zeroes) data. Must be zeroed during C-startup. Must be placed in the compiler SDA (near addressing).
`os_pur`	RAM	RTA-OSEK uninitialized data. Zeroed by StartOS().
`os_pur2`	RAM	RTA-OSEK uninitialized data. Must be zeroed during C-startup.
`os_pnur`	RAM	RTA-OSEK near uninitialized data. Zeroed by StartOS(). Must be placed in the compiler SDA (near addressing)
`os_pnur2`	RAM	RTA-OSEK near uninitialized data. Must be zeroed during C-startup. Must be placed in the compiler SDA (near addressing)
`os_trace_ram`	RAM	RTA-TRACE buffer. RTA-TRACE buffer. Can be located in 'far' RAM. Does not need to be initialized.
`os_trace_ram_bss`	RAM	RTA-TRACE buffer. RTA-TRACE buffer. Can be located in 'far' RAM. Does not need to be initialized.

The RTA-OSEK example application contains a valid linker file `example.ld`, containing the necessary sections and section attributes.

2.4 Debugger

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

ORTI compatible debuggers	Lauterbach TRACE32

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

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for MPC55xx/GreenHills. You can find out more about configuring interrupts for RTA-OSEK in the *RTA-OSEK User Guide*.

3.1.1 Interrupt Levels

In RTA-OSEK interrupts are allocated an Interrupt Priority Level (IPL). This is a processor independent abstraction of the interrupt priorities that are available on the target hardware. You can find out more about IPLs in the *RTA-OSEK User Guide*. The hardware interrupt controller is explained in the *MPC5554 Reference Manual and the MPC5554 e200z6 Core Supplementary Reference Manual*.

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

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

3.1.2 Interrupt Vectors

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

Vector	Legality
0x0 to 0x22	The CPU vectors only handle Category 1 ISRs
0x10000 to the maximum INTC vector for the chip variant in 0x10 steps	The INTC vectors can handle Category 1 and 2 ISRs

The valid base addresses for the vector table are:

Base Address	Notes
IVPR	The base address of the vector table should be aligned to a 4 or 64Kbyte 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 Green Hills Software, Inc. C compiler can generate appropriate interrupt handling code for a C function decorated with the `__interrupt` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

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

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

The following table shows the syntax for labels attached to RTA-OSEK Category 2 interrupt handlers (VVVVV represents the 5 hex digit, upper-case, zero-padded value of the vector location).

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

Note that in the RTA-OSEK GUI and `.oil` files, the INTC vectors begin at `0x10000`. This is only a notational convenience to distinguish between CPU vectors and INTC vectors. Only the last four hexadecimal digits are used as offsets in target code.

3.1.6 Processor Mode

RTA-OSEK operates in the processor's supervisor mode and expects that applications shall operate in supervisor mode.

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

3.1.7 INTC and CPU Vector Offset Mapping

The MPC55XX has two exception sources: the CPU and the Interrupt Controller. Handling of INTC interrupts may be configured in "hardware" or "software" vector mode.

RTA-OSEK directly supports hardware vector mode, explained in this section. If you require software vector mode, some guidance can be found in the next section.

On receipt of an INTC interrupt in hardware vector mode, the CPU jumps to the address formed by adding IVPR to the hard-wired offset for that particular interrupt. The least significant 4 bits of this address are always 0, so all INTC handlers must start on a 16-byte boundary.

RTA-OSEK emits the array `os_INTC_vectors`, representing the INTC vector table. This is a contiguous block of 16-byte elements on single core variants. When configured, the element contains a branch to the OS entry-point for that ISR and 12 bytes of padding. Dual core variants use a contiguous block of 4-byte elements with no padding.

Because the IVPR supplies either the most significant 16 or 20 bits (depending upon core type) of the address, `os_INTC_vectors` must be aligned to a 64 or 4KB boundary respectively. In other words:

- `(os_INTC_vectors & 0x0000ffff) == 0` or
- `(os_INTC_vectors & 0x00000fff) == 0`
- `os_INTC_vectors == IVPR`

RTA-OSEK places `os_INTC_vectors` at the start of section `os_intvec` to facilitate this.

For CPU interrupts and exceptions, the vector address is programmable. Each interrupt source has a dedicated offset register IVOR*x*. On receipt of an interrupt the CPU jumps to the address in the IVOR*x* register for that particular interrupt source, relative to the IVPR.

RTA-OSEK emits an array `os_CPU_vectors`. This is a contiguous block of 4-byte elements. Where a CPU interrupt is configured, the element contains a branch to the OS entry-point for that interrupt.

Because CPU interrupt handler addresses are also relative to IVPR, `os_CPU_vectors` must be located in the same 64KB block of memory as os_INTC_vectors. RTA-OSEK places os_CPU_vectors directly after os_INTC_vectors in section os_intvec to facilitate this.

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

Important: In the RTA-OSEK GUI, INTC vectors begin at 0x10000. This is only a naming convention for the GUI, and only the last 4 digits are used as an offset.

3.1.8 Using Software Vector Mode

RTA-OSEK directly supports Hardware Vector Mode.

While Software Vector Mode is not directly supported by RTA-OSEK, some expert users may judge that software vector mode is desirable. Software Vector Mode increases the entry time into Category 1 and 2 ISRs but may save memory on the INTC vector table.

When an INTC interrupt is received in Software Vector Mode, instead of the CPU jumping to IVPR + *offset*, the INTC uses *offset* to write an address into INTC_IACKR and the CPU jumps to address IVPR + IVOR4. Consult the hardware reference manual for the exact details of how this address is generated.

The user must implement a software demultiplexer (demux) that reads the INTC_IACKR register to identify the interrupt that triggered, and enter the relevant handler routine. For interrupts configured as Category 2 interrupts, this means entering the kernel at a label of the form os_wrapper_10000.

Because the demux is accessed using IVPR it must be located in the same 64KB block as the os_CPU_vectors table.

One such demux might simply use table of entry-point addresses, thus saving 12 bytes-per-entry compared to Hardware Vector Mode on a single core variant.

Important: Because RTA-OSEK is built to directly support Hardware Vector Mode, you cannot simply call the operating system entry-point. The following paragraphs explain the calling convention.

Because the RTA-OSEK kernel expects to be entered as an interrupt handler, the flow of control terminates with an rfi instruction. Because the demux function requires some stack it is important that this does NOT result in an actual return from interrupt, but that it returns to demux to allow the stack to be tidied up. From the BookE reference manual, we see that rfi simply writes SRR1 to the MSR and jumps to SSR0. Thus our calling convention is as follows:

- Preserve SRR0, SRR1 and MSR on the stack
- Place our return address in SRR0. **Do not** write MSR to SRR1
- Branch without link to the entry-point for the interrupt.
- Restore MSR from the stack
- Restore SRR1 and SRR0 from the stack

The demux itself is an interrupt handler, so it must preserve all registers it uses and terminate in rfi.

Below is an example that reads INTC_IACKR and branches to an address taken from a vector table.

```
interrupt_demux:

  stwu    r1,-32(r1)          ; Allocate some stack
  stw     r3,8(r1)            ; Preserve r3
  mfspr   r3,ctr               ; Preserve CTR
  stw     r3,12(r1)
  mfssr0  r3                   ; Preserve SSR0
  stw     r3,16(r1)
  mfssr1  r3                   ; Preserve SSR1
  stw     r3,20(r1)
  mfmsr   r3                   ; Preserve MSR
  stw     r3,24(r1)

  ; Since the OS uses rfi when finished, we set up
  ; a phony interrupt context that returns to this
  ; function.  This enables us to tidy our stack
  ; before really rfi.

  ; ssr0 <- return address
  lis     r3,%hi(demux_returnaddr)
  ori     r3,r3,%lo(demux_returnaddr)
  mtssr0  r3

  ; Now we find the entrypoint for our ISR by
  ; whatever method we prefer.  This example uses a
  ; vector table based at VTBA.

  ; Get address of INTC_ACKR into r3
  lis     r3,%hi(0xFFF48010)
  ori     r3,r3,%lo(0xFFF48010)

  ; Now load the value from INTC_ACKR, which also
  ; clears interrupt line from INTC to CPU
```

```
  lwz     r3,0x0(r3)

  ; Value at INTC_ACKR was pointer into vector
  ; table, get the vector (=ISR entry point)
  lwz     r3,0x0(r3)

  ; Move ISR entry point to ctr
  mtctr   r3

  ; Indirect branch to ISR entry point,
  ; e.g. os_wrapper_10000
  bcctr   20,0

demux_returnaddr:
  ; This is where we return when the rfi in the
  ; OS is encountered.
  ; NOTE: at this point interrupts are enabled.
  ; If another interrupt (at any level) is
  ; pending, it will be serviced BEFORE we
  ; can release the stack.
  ; Applications that saturate the interrupts
  ; will overflow the stack.
  lwz     r3,24(r1)           ; Restore MSR. Note:
  mtmsr   r3                  ; disables interrupts
                              ; until rfi
  lwz     r3,20(r1)           ; Restore SSR1
  mrssr1  r3
  lwz     r3,16(r1)           ; Restore SSR0
  mrssr1  r3
  lwz     r3,12(r1)           ; Restore CTR
  mtctr   r3
  lwz     r3,8(r1)            ; Restore r3
  addi    r1,r1,32            ; Restore the SP

  ; Now we really return from interrupt.
  rfi
```

The above example does not use `os_INTC_vectors`. It uses a custom vector table based around `INTC_IACKR`. The custom vector table consists of 4-byte addresses of the kernel entry-points. The table itself must be located on a 2KB boundary and the base address written to INTC_IACKR.

Important: The stack measurements given in the binding manual and built into the toolchain are based on Hardware Vector Mode. If you implement a demux function, you must take into account any additional stack used by it, or stack faults may occur.

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:

  ; Save interrupt context following the EABI
  ; Preserve scratch register
  stw     r0,8(r1)

  ; SRR0,1 contain ISR return context.
  mfspr   r0,srr0

  ; Push it to allow nested interrupts
  stw     r0,12(r1)
  mfspr   r0,srr1
  stw     r0,16(r1)

  ; Increment the interrupt counter before enabling
  ; global interrupts
  stw     r4,44(r1)       ; Preserve R4
  lwz     r4,%sdaoff(os_isr_count)(r13)
  addi    r4,r4,1
  dtw     r4,%sdaoff(os_isr_count)(r13)

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

  ; Cat 1 blocking ends here

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

#ifdef OS_STANDARD_BUILD
  ; Set up function addr for the indirect branch
  mtspr   ctr,r3
#endif /* OS_STANDARD_BUILD */
```

```
  ; R3 - already done in the outer wrapper
  ; stw   r3,40(r1)

  ; R4 - taken care of above
  ; stw   r4,44(r1)
  stw     r5,48(r1)
  stw     r6,52(r1)
  stw     r7,56(r1)
  stw     r8,60(r1)
  stw     r9,64(r1)
  stw     r10,68(r1)
  stw     r11,72(r1)
  stw     r12,76(r1)

#ifdef OS_STANDARD_BUILD
  ; Indirect branch to ISR function
  bcctrl  20,0
#endif /* OS_STANDARD_BUILD */

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

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

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

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

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

  ; Return the IPL level to that before the
  ; interrupt triggered
  ; INTC_CPR is at address 0xFFF48008
  addis   r4,r0,%hiadj(0xFFF48008)
```

```
stw     r3,%lo(0xFFF48008)(r4)

; Decrement the interrupt counter after enabling
; global interrupts
lwz     r4,%sdaoff(os_isr_count)(r13)
addi    r4,r4,-1
stw     r4,%sdaoff(os_isr_count)(r13)

lwz     r4,44(r1)
lwz     r3,40(r1)

; Restore the remaining context
lwz     r0,16(r1)       ; Restore SRR0/1
mtspr   srr1,r0
lwz     r0,12(r1)
mtspr   srr0,r0
lwz     r0,8(r1)

addi    r1,r1,80        ; Restore the SP

; Return from interrupt - Cat 1 blocking ends
rfi
```

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 cores directly supported are the e200z1 (MPC5514, MPC5516), e200z3 (MPC5534, SPC563M), e200z4 (MPC5643L, MPC5644A, MPC5645B, MPC5645C, MPC5646B, MPC5646C, SPC564A), e200z6 (MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, MPC5567), e200z7 (MPC5674F, MPC5675K) and MPC55xx Generic. Further variants can be supported by contacting ETAS.

When RTA-OSEK generates an interrupt vector table for the MPC55xx INTC interrupts, it only emits data for interrupts 0x10000 up to the *highest declared interrupt*. This allows RTA-OSEK to cope efficiently with chip variants with different 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 INTC interrupts declared in the application (from 0x10000 to the highest declared interrupt). The array is terminated by a byte value 0xFF. The example application demonstrates a method of setting the INTC_PSR registers using this array.

If vector table generation is disabled in RTA-OSEK then the array is not assembled by default. To force assembly of the array, define the preprocessor symbol `OS_GEN_PSC_TABLE`

3.1.11 INTC Race Condition

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

When the interrupt controller is configured to use hardware interrupt vectors the manipulation of this counter is handled by the RTA-OSEK kernel libraries. If the interrupt controller is configured to use software interrupt vectors then `os_isr_counter` must be incremented at the start of the interrupt handler before global interrupts are re-enabled (as demonstrated in the code for `os_mid_wrapper` in section 3.1.8). At the end of the interrupt handler `os_isr_counter` must be decremented (again as demonstrated in `os_mid_wrapper`).

3.1.12 OS_LIFO_LOAD

The RTA-OSEK Component for the MPC55XX/GreenHills supports the macro OS_LIFO_LOAD. It can be used by writers of common interrupt entry functions to push a value onto the INTC LIFO. Refer to the comments in `ostarget.h` for further details.

3.1.13 Default Interrupt

RTA-OSEK allows you to define a ‘default interrupt’ handler to catch unexpected interrupts. All unused interrupts have their interrupt vectors directed to the specified entrypoint. This code must correctly handle the interrupt context, in the same way as a Category 1 ISR.

Because RTA-OSEK for MPC55xx only emits interrupt vectors 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.

3.2 Register Settings

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

The RTA-OSEK Component does not reserve the use of any hardware registers.

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: the stack pointer R1 is adjusted only once in each function, a back-link is maintained, and the stack pointer is always aligned to a 16-byte boundary.

RTA-OSEK expects the stack to be located in the default stack section `stack`, which must be placed in internal or external RAM.

The application startup code must set R1 to a suitable value. The Green Hills linker automatically generates symbol `__ghsend_stack` which contains the correct initial value for R1.

3.4 Floating point

The Freescale PowerPC book E CPUs contains 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 on the SPE:

All SPE operations operate directly upon the GPRs. The GPRs are in fact 64 bits wide, but only the Vector instructions access the upper 32 bits. All non-SPE instructions and the Floating Point SPE instructions use only the lower 32 bits of the GPRs. If you intend to use multiply-and-accumulate (MAC) in your application, be sure to read the note later in this section.

RTA-OSEK provides "floating point wrappers" to save and restore the additional context required when using Floating Point or Vector instructions. To enable this functionality, use the RTA-OSEK GUI to declare the relevant tasks and/or category 2 ISRs as "floating point".

When a task or category 2 ISR is declared as floating point, the kernel uses the functions in *<RTA-OSEK location>*`\GHS55xx\inc\osfptgt.c` to store additional context. The supplied `osfptgt.c` supports a lean version for applications that only use 32-bit registers, and a complete version for applications that use 64-bit registers. The 64-bit version is selected by defining the preprocessor symbol `OS_SPE_VECTOR`.

If your application uses only integer and floating point instructions, not Vector or MAC, then build `osfptgt.c` with `OS_SPE_VECTOR` undefined. The floating-point wrapper will store only the `SPEFSCR`. This is the most lightweight approach.

If your application uses MAC instructions or Vector instructions, then build `osfptgt.c` with `OS_SPE_VECTOR` defined. The floating-point wrapper will store `SPEFSCR`, all the 64-bit registers, and the Floating Point Accumulator.

Note: If your application uses MAC instructions but not Vector instructions, you should build `osfptgt.c` with `OS_SPE_VECTOR` defined, since the reset, save, and restore of the floating point accumulator require the use of a 64-bit register. However, it is likely that you do not need all the 64-bit registers to be protected by `osfptgt` in this way. If you are able to define and limit which registers you use in their 64-bit versions then it may be worthwhile to write a custom `osfptgt.c` that is leaner than the full 64-bit version supplied.

Note: The performance figures have been collected for a system that does not use the SPE.

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	**MPC5554**
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	158	158	162	282	282	286
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	230	230	234	354	354	358
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	288	288	292	412	412	416
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	52	52	52	52	52	52
Category 2 ISR, floating-point	RAM	4	4	4	4	4	4
	ROM	92	92	92	92	92	92
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	52	52	52	52	52	52
Counter	RAM	4	4	4	4	4	4
	ROM	104	104	104	104	104	104
Message	RAM	11	11	11	31	31	31
	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 Events Shared Task Priorities Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		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	140	140	140	140	140	140
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 Events Shared Task Priorities Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		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	128	128	128	128	128	128
Category 2 ISR, floating-point	RAM	16	16	16	16	16	16
	ROM	136	136	136	136	136	136
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	52	52	52	52	52	52
Counter	RAM	4	4	4	4	4	4
	ROM	104	104	104	104	104	104
Message	RAM	11	11	11	31	31	31
	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	140	140	140	140	140	140
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	140	140	140	140	140	140
Category 2 ISR, floating-point	RAM	20	20	20	20	20	20
	ROM	148	148	148	148	148	148
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	56	56	56	56	56	56
Counter	RAM	4	4	4	4	4	4
	ROM	108	108	108	108	108	108
Message	RAM	11	11	11	31	31	31
	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	140	140	140	140	140	140
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	180	272	328	188	280	364
	NS		156	248	296	164	256	340
	KL	2	104	196	244	112	204	276
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	20	20	20	20	20	20
ChainTask	SWL	1, 8	168	260	316	176	268	356
	SWH	1, 9	236	340	396	244	348	440
	NSL	8	168	260	316	176	268	356
	NSH	9	220	324	380	228	332	416
Schedule			140	140	200	140	140	200
GetTaskID			40	40	40	40	40	40
GetTaskState			100	100	100	124	124	124
EnableAllInterrupts			36	36	36	36	36	36
DisableAllInterrupts			52	52	52	52	52	52
ResumeAllInterrupts			52	52	52	52	52	52
SuspendAllInterrupts			76	76	76	76	76	76
ResumeOSInterrupts			48	48	48	48	48	48
SuspendOSInterrupts			80	80	80	80	80	80
GetResource	Task	7	36	36	44	36	36	44
	Combined	6	116	116	116	116	116	116
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	116	116	116	116	116	116
	Combined	6	296	296	296	296	296	296
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	168	168	288
	NS		n/a	n/a	n/a	104	104	224
	NS1i	10	n/a	n/a	n/a	64	n/a	n/a
	KL	2	n/a	n/a	n/a	72	72	176
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	64	64	64

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	336	336	592
	fp	11	n/a	n/a	n/a	376	376	664
	1i	10	n/a	n/a	n/a	28	n/a	n/a
GetAlarmBase			68	68	68	68	68	68
GetAlarm			156	156	156	156	156	156
SetRelAlarm			700	700	700	700	700	700
SetAbsAlarm			748	748	748	748	748	748
CancelAlarm			144	144	144	144	144	144
InitCounter			68	68	68	68	68	68
GetCounterValue			76	76	76	76	76	76
GetScheduleTableStatus		34	108	144	144	108	144	144
NextScheduleTable		34	148	308	308	148	308	308
StartScheduleTable		34	200	288	288	200	288	288
StopScheduleTable		34	152	204	204	152	204	204
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	16	16	16
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		76	76	76	76	76	76
GetISRID		4	n/a	n/a	n/a	n/a	n/a	n/a
Process container	Yielding	32	60	60	60	60	60	60
Process container	Non-Yielding	33	40	40	40	40	40	40
osek_tick_alarm	<default>		100	100	100	100	100	100
	KL	2	68	68	68	68	68	68
osek_incr_counter			60	60	60	60	60	60
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			232	232	232	232	232	232
ShutdownOS	NoHook	12	40	40	40	40	40	40
	Hook	13	56	56	56	56	56	56
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			56	56	56	56	56	56
StopCOM			28	28	28	28	28	28
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	120	120	120	308	308	308

Configuration / Events / Shared Task Priorities / Multiple Task Activations			Application Uses: No / Events No / STP No / MTA No	No / No / Yes	No / Yes	Yes / No / No	Yes / No / Yes	Yes / Yes
	CCCB	15	308	308	308	308	308	308
GetMessageResource			84	84	84	84	84	84
ReleaseMessageResource			72	72	72	72	72	72
GetMessageStatus			88	88	88	88	88	88
SendMessage	SW CCCA	1, 14	160	160	160	492	492	492
	SW CCCB	1, 15	456	456	456	492	492	492
	NS CCCA	14	160	160	160	492	492	492
	NS CCCB	15	456	456	456	492	492	492
	KL CCCA	2, 14	100	100	100	404	404	404
	KL CCCB	2, 15	384	384	384	404	404	404
main_dispatch	NoHook	12	180	180	236	180	180	236
	Hook	13	224	224	280	224	224	280
sub_dispatch	B1LF	19	52	52	52	52	52	52
	B1HI	20	140	140	140	140	140	140
	B1HF	21	140	140	140	140	140	140
	B2LI	22	n/a	108	144	n/a	108	144
	B2LF	23	n/a	116	152	n/a	116	152
	B2HI	24	n/a	348	420	n/a	348	420
	B2HF	25	n/a	340	428	n/a	340	428
	E1HI	26	n/a	n/a	n/a	444	444	536
	E1HF	27	n/a	n/a	n/a	452	452	544
	E2HI	28	n/a	n/a	n/a	n/a	n/a	536
	E2HF	29	n/a	n/a	n/a	n/a	n/a	544
ErrorHook support		16	76	76	76	76	76	76
	ServiceID	17	84	84	84	84	84	84
	Parameters	18	104	104	104	104	104	104
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	176	356	420	212	368	444
	NS		152	332	396	188	344	420
	KL	2	100	280	344	132	308	380
ChainTaskset	SWL	1, 8	172	360	424	180	392	452
	SWH	1, 9	224	412	452	232	420	496
	NSL	8	172	360	424	180	392	452
	NSH	9	216	404	444	224	412	488
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			56	56	56	56	56	56
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			56	56	56	56	56	56
TestSubTaskset			68	68	68	68	68	68
TestEquivalentTaskset			64	64	64	64	64	64
TickSchedule	SW	1	316	312	312	312	312	312
	NS		288	272	272	272	272	272
	KL	2	240	256	256	256	256	256
AdvanceSchedule	SW	1	272	272	272	272	272	272
	NS		244	244	244	244	244	244
	KL	2	228	236	236	236	236	236
StartSchedule			156	156	156	156	156	156
StopSchedule			116	116	116	116	116	116
GetScheduleStatus			180	180	180	180	180	180
GetScheduleValue			144	144	144	144	144	144
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			52	52	52	52	52	52
GetExecutionTime			8	8	8	8	8	8
GetLargestExecutionTime			16	16	16	16	16	16
ResetLargestExecutionTime			8	8	8	8	8	8
GetStackOffset			60	60	60	60	60	60

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	180	272	328	188	280	364
	NS		156	248	296	164	256	340
	KL	2	104	196	244	112	204	276
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	20	20	20	20	20	20
ChainTask	SWL	1, 8	168	260	316	176	268	356
	SWH	1, 9	236	340	396	244	348	440
	NSL	8	168	260	316	176	268	356
	NSH	9	220	324	380	228	332	416
Schedule			164	164	224	164	164	224
GetTaskID			40	40	40	40	40	40
GetTaskState			100	100	100	124	124	124
EnableAllInterrupts			36	36	36	36	36	36
DisableAllInterrupts			52	52	52	52	52	52
ResumeAllInterrupts			52	52	52	52	52	52
SuspendAllInterrupts			76	76	76	76	76	76
ResumeOSInterrupts			48	48	48	48	48	48
SuspendOSInterrupts			80	80	80	80	80	80
GetResource	Task	7	36	36	44	36	36	44
	Combined	6	116	116	116	116	116	116
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	140	140	140	140	140	140
	Combined	6	360	360	360	360	360	360
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	168	168	288
	NS		n/a	n/a	n/a	104	104	224
	NS1i	10	n/a	n/a	n/a	64	n/a	n/a
	KL	2	n/a	n/a	n/a	72	72	176
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	64	64	64
GetEvent			n/a	n/a	n/a	20	20	20
WaitEvent	<default>		n/a	n/a	n/a	408	408	652
	fp	11	n/a	n/a	n/a	444	444	724

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	144	n/a	n/a
GetAlarmBase			68	68	68	68	68	68
GetAlarm			156	156	156	156	156	156
SetRelAlarm			700	700	700	700	700	700
SetAbsAlarm			748	748	748	748	748	748
CancelAlarm			144	144	144	144	144	144
InitCounter			68	68	68	68	68	68
GetCounterValue			76	76	76	76	76	76
GetScheduleTableStatus		34	108	144	144	108	144	144
NextScheduleTable		34	148	308	308	148	308	308
StartScheduleTable		34	200	288	288	200	288	288
StopScheduleTable		34	152	204	204	152	204	204
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	16	16	16
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		76	76	76	76	76	76
GetISRID		4	52	52	52	52	52	52
Process container	Yielding	32	60	60	60	60	60	60
Process container	Non-Yielding	33	40	40	40	40	40	40
osek_tick_alarm	<default>		100	100	100	100	100	100
	KL	2	68	68	68	68	68	68
osek_incr_counter			60	60	60	60	60	60
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			284	284	284	284	284	284
ShutdownOS	NoHook	12	40	40	40	40	40	40
	Hook	13	56	56	56	56	56	56
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			56	56	56	56	56	56
StopCOM			28	28	28	28	28	28
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	120	120	120	308	308	308
	CCCB	15	308	308	308	308	308	308
GetMessageResource			84	84	84	84	84	84
ReleaseMessageResource			72	72	72	72	72	72

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetMessageStatus			88	88	88	88	88	88
SendMessage	SW CCCA	1, 14	160	160	160	492	492	492
	SW CCCB	1, 15	456	456	456	492	492	492
	NS CCCA	14	160	160	160	492	492	492
	NS CCCB	15	456	456	456	492	492	492
	KL CCCA	2, 14	100	100	100	404	404	404
	KL CCCB	2, 15	384	384	384	404	404	404
main_dispatch	NoHook	12	236	236	288	236	236	288
	Hook	13	280	280	332	280	280	332
sub_dispatch	B1LF	19	40	40	40	40	40	40
	B1HI	20	124	124	124	124	124	124
	B1HF	21	132	132	132	132	132	132
	B2LI	22	n/a	80	116	n/a	80	116
	B2LF	23	n/a	88	124	n/a	88	124
	B2HI	24	n/a	288	392	n/a	288	392
	B2HF	25	n/a	296	392	n/a	296	392
	E1HI	26	n/a	n/a	n/a	472	472	564
	E1HF	27	n/a	n/a	n/a	480	480	572
	E2HI	28	n/a	n/a	n/a	n/a	n/a	564
	E2HF	29	n/a	n/a	n/a	n/a	n/a	572
ErrorHook support		16	76	76	76	76	76	76
	ServiceID	17	84	84	84	84	84	84
	Parameters	18	104	104	104	104	104	104
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	116	116	116	116	116	116
Timing_termination		4	128	128	128	128	128	128
ActivateTaskset	SW	1	176	356	420	212	368	444
	NS		152	332	396	188	344	420
	KL	2	100	280	344	132	308	380
ChainTaskset	SWL	1, 8	172	360	424	180	392	452
	SWH	1, 9	224	412	452	232	420	496
	NSL	8	172	360	424	180	392	452
	NSH	9	216	404	444	224	412	488
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			56	56	56	56	56	56
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			56	56	56	56	56	56

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestSubTaskset			68	68	68	68	68	68
TestEquivalentTaskset			64	64	64	64	64	64
TickSchedule	SW	1	316	312	312	312	312	312
	NS		288	272	272	272	272	272
	KL	2	240	256	256	256	256	256
AdvanceSchedule	SW	1	272	272	272	272	272	272
	NS		244	244	244	244	244	244
	KL	2	228	236	236	236	236	236
StartSchedule			156	156	156	156	156	156
StopSchedule			116	116	116	116	116	116
GetScheduleStatus			180	180	180	180	180	180
GetScheduleValue			144	144	144	144	144	144
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			52	52	52	52	52	52
GetExecutionTime			148	148	148	148	148	148
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			24	24	24	24	24	24
GetStackOffset			60	60	60	60	60	60

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	432	524	564	440	532	600
	NS		572	612	644	572	612	688
	KL	2	328	412	456	332	424	492
TerminateTask	LExt	3	236	236	236	236	236	236

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	316	316	316	316	316	316
ChainTask	SWL	1, 8	588	684	728	596	692	768
	SWH	1, 9	740	804	868	748	832	900
	NSL	8	672	768	812	680	776	828
	NSH	9	796	880	924	824	888	956
Schedule			388	388	448	388	388	448
GetTaskID			60	60	60	60	60	60
GetTaskState			376	376	376	344	344	344
EnableAllInterrupts			56	56	56	56	56	56
DisableAllInterrupts			72	72	72	72	72	72
ResumeAllInterrupts			140	140	140	140	140	140
SuspendAllInterrupts			96	96	96	96	96	96
ResumeOSInterrupts			136	136	136	136	136	136
SuspendOSInterrupts			100	100	100	100	100	100
GetResource	Task	7	596	596	560	596	596	560
	Combined	6	580	580	580	580	580	580
	CLEx	3	468	468	468	468	468	468
ReleaseResource	Task	7	580	580	580	580	580	580
	Combined	6	684	684	684	684	684	684
	CLEx	3	536	536	536	536	536	536
SetEvent	SW	1	n/a	n/a	n/a	536	536	644
	NS		n/a	n/a	n/a	604	604	712
	NS1i	10	n/a	n/a	n/a	416	n/a	n/a
	KL	2	n/a	n/a	n/a	416	416	528
	KL1i	2, 10	n/a	n/a	n/a	336	n/a	n/a
ClearEvent			n/a	n/a	n/a	216	216	216
GetEvent			n/a	n/a	n/a	328	328	328
WaitEvent	<default>		n/a	n/a	n/a	632	632	828
	fp	11	n/a	n/a	n/a	668	668	900
	1i	10	n/a	n/a	n/a	392	n/a	n/a
GetAlarmBase			248	248	248	248	248	248
GetAlarm			272	272	272	272	272	272
SetRelAlarm			952	952	952	952	952	952
SetAbsAlarm			1000	1000	1000	1000	1000	1000
CancelAlarm			252	252	252	252	252	252
InitCounter			356	356	356	356	356	356
GetCounterValue			256	256	256	256	256	256

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	132	168	168	132	168	168
NextScheduleTable		34	168	344	344	168	344	344
StartScheduleTable		34	228	324	324	228	324	324
StopScheduleTable		34	180	240	240	180	240	240
ScheduleTable expiry point	ActivateTask		12	12	12	12	12	12
ScheduleTable expiry point	SetEvent		n/a	n/a	n/a	16	16	16
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		76	76	76	76	76	76
GetISRID		4	72	72	72	72	72	72
Process container	Yielding	32	60	60	60	60	60	60
Process container	Non-Yielding	33	40	40	40	40	40	40
osek_tick_alarm	<default>		180	180	180	180	180	180
	KL	2	68	68	68	68	68	68
osek_incr_counter			60	60	60	60	60	60
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			312	312	312	312	312	312
ShutdownOS	NoHook	12	52	52	52	52	52	52
	Hook	13	68	68	68	68	68	68
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			84	84	84	84	84	84
StopCOM			72	72	72	72	72	72
ReadFlag			40	40	40	40	40	40
ResetFlag			44	44	44	44	44	44
ReceiveMessage	CCCA	14	272	272	272	504	504	504
	CCCB	15	504	504	504	504	504	504
GetMessageResource			176	176	176	176	176	176
ReleaseMessageResource			176	176	176	176	176	176
GetMessageStatus			216	216	216	216	216	216
SendMessage	SW CCCA	1, 14	308	308	308	652	652	652
	SW CCCB	1, 15	632	632	632	652	652	652
	NS CCCA	14	308	308	308	652	652	652
	NS CCCB	15	632	632	632	652	652	652
	KL CCCA	2, 14	256	256	256	552	552	552
	KL CCCB	2, 15	532	532	532	552	552	552
main_dispatch	NoHook	12	236	236	288	236	236	288

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	280	280	332	280	280	332
sub_dispatch	B1LF	19	40	40	40	40	40	40
	B1HI	20	124	124	124	124	124	124
	B1HF	21	132	132	132	132	132	132
	B2LI	22	n/a	80	116	n/a	80	116
	B2LF	23	n/a	88	124	n/a	88	124
	B2HI	24	n/a	288	392	n/a	288	392
	B2HF	25	n/a	296	392	n/a	296	392
	E1HI	26	n/a	n/a	n/a	472	472	564
	E1HF	27	n/a	n/a	n/a	480	480	572
	E2HI	28	n/a	n/a	n/a	n/a	n/a	564
	E2HF	29	n/a	n/a	n/a	n/a	n/a	572
ErrorHook support		16	188	188	188	188	188	188
	ServiceID	17	196	196	196	196	196	196
	Parameters	18	212	212	212	212	212	212
validity_checks		3	48	48	48	48	48	48
Timing_dispatch		4	116	116	116	116	116	116
Timing_termination		4	128	128	128	128	128	128
ActivateTaskset	SW	1	504	592	652	544	616	700
	NS		564	660	720	604	684	768
	KL	2	364	476	536	404	476	560
ChainTaskset	SWL	1, 8	664	736	796	676	756	840
	SWH	1, 9	768	864	924	824	884	968
	NSL	8	716	812	872	752	832	916
	NSH	9	836	932	992	872	952	1036
GetTasksetRef			256	256	256	256	256	256
MergeTaskset			396	396	396	396	396	396
AssignTaskset			264	264	264	264	264	264
RemoveTaskset			392	392	392	392	392	392
TestSubTaskset			376	376	376	376	376	376
TestEquivalentTaskset			372	372	372	372	372	372
TickSchedule	SW	1	508	480	480	480	480	480
	NS		592	536	536	536	536	536
	KL	2	476	456	456	456	456	456
AdvanceSchedule	SW	1	536	496	496	496	496	496
	NS		620	568	568	568	568	568
	KL	2	488	448	448	448	448	448

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			472	472	472	472	472	472
StopSchedule			284	284	284	284	284	284
GetScheduleStatus			344	344	344	344	344	344
GetScheduleValue			268	268	268	268	268	268
GetScheduleNext			136	136	136	136	136	136
SetScheduleNext			328	328	328	328	328	328
GetArrivalpointDelay			212	212	212	212	212	212
SetArrivalpointDelay			288	288	288	288	288	288
GetArrivalpointTasksetRef			212	212	212	212	212	212
GetArrivalpointNext			212	212	212	212	212	212
SetArrivalpointNext			332	332	332	332	332	332
TestArrivalpointWritable			232	232	232	232	232	232
GetExecutionTime			216	216	216	216	216	216
GetLargestExecutionTime			180	180	180	180	180	180
ResetLargestExecutionTime			164	164	164	164	164	164
GetStackOffset			60	60	60	60	60	60

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	119	152	189	122	154	212
	NS	110	143	178	108	135	198
	KL	55	93	124	72	94	147
TerminateTask	LExt	0	0	0	0	0	0
	H	239	241	245	239	240	251
ChainTask	SWL	388	437	517	453	492	569
	SWH	497	553	618	554	590	683
	NSL	381	441	508	460	489	572
	NSH	485	545	610	556	593	685
Schedule	SW	123	121	128	112	112	128
GetTaskID		30	29	30	39	39	35
GetTaskState		88	91	86	100	101	104
EnableAllInterrupts		36	34	38	33	33	35
DisableAllInterrupts		50	47	47	42	42	46
ResumeAllInterrupts		49	49	49	45	45	46
SuspendAllInterrupts		61	62	62	70	70	65
ResumeOSInterrupts		50	50	50	45	45	46
SuspendOSInterrupts		62	63	63	71	71	66
GetResource	Task	46	46	44	46	45	47
	Combined	87	90	90	90	90	87
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	101	100	102	104	104	104
	Combined	162	163	152	155	155	165
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	130	129	143
	NS	n/a	n/a	n/a	88	90	101
	KL	n/a	n/a	n/a	51	58	55
ClearEvent		n/a	n/a	n/a	67	67	68
GetEvent		n/a	n/a	n/a	27	27	26
WaitEvent	<default>	n/a	n/a	n/a	693	704	792
	fp	n/a	n/a	n/a	697	706	805
GetAlarmBase		85	82	85	79	78	75
GetAlarm		129	127	127	118	120	122

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		159	158	158	158	155	157
SetAbsAlarm		160	159	153	155	148	154
CancelAlarm		105	100	105	106	109	104
InitCounter		69	69	69	62	65	64
GetCounterValue		69	69	69	67	70	71
osek_tick_alarm	<default>	88	84	87	86	92	85
	KL	49	48	46	52	53	51
osek_incr_counter		15	15	15	14	14	14
GetActiveApplicationMode		8	8	8	8	8	8
StartOS		1535	1101	1276	1417	1553	1243
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	53	53	53	55	55	53
InitCOM		14	10	14	8	8	7
CloseCOM		15	13	15	14	14	13
StartCOM		62	57	62	154	156	152
StopCOM		22	24	22	22	22	20
ReadFlag		n/a	n/a	n/a	14	14	14
ResetFlag		n/a	n/a	n/a	9	9	9
ReceiveMessage		96	95	96	338	340	319
GetMessageResource		n/a	n/a	n/a	116	116	123
ReleaseMessageResource		n/a	n/a	n/a	160	160	166
GetMessageStatus		n/a	n/a	n/a	67	67	66
SendMessage	SW	242	274	312	467	508	551
	NS	227	259	295	467	488	551
	KL	119	162	190	370	402	442
ActivateTaskset	SW	108	720	820	117	561	755
	NS	96	677	778	99	698	798
	KL	45	661	698	54	644	708
	SW2	108	720	820	117	561	755
	NS2	96	677	778	99	698	798
	KL2	45	661	698	54	644	708
ChainTaskset	SWL	374	820	937	442	1052	1121
	SWH	485	1104	1032	553	1144	1306
	NSL	369	1005	1088	432	1011	1211
	NSH	486	1070	1214	543	1132	1204
GetTasksetRef		27	25	28	26	27	27

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		63	60	65	61	62	63
AssignTaskset		24	22	22	21	21	21
RemoveTaskset		63	59	60	59	57	62
TestSubTaskset		67	66	67	68	66	66
TestEquivalentTaskset		60	66	59	67	63	59
TickSchedule	SW	180	856	886	240	851	904
	NS	171	846	875	226	839	889
	KL	115	794	838	189	802	847
	SW2	180	856	883	240	834	896
	NS2	171	846	874	226	822	881
	KL2	113	792	835	188	784	838
AdvanceSchedule	SW	170	833	877	228	832	884
	NS	152	828	856	210	817	879
	KL	106	789	827	181	787	844
	SW2	167	836	877	228	816	876
	NS2	149	829	856	210	801	871
	KL2	104	791	828	180	773	835
StartSchedule		124	125	121	124	122	124
StopSchedule		109	111	113	116	112	114
GetScheduleStatus		121	117	123	118	122	120
GetScheduleValue		120	123	116	121	121	123
GetScheduleNext		29	28	29	26	26	25
SetScheduleNext		27	28	26	21	21	23
GetArrivalpointDelay		23	22	23	25	25	23
SetArrivalpointDelay		20	18	20	23	23	22
GetArrivalpointTasksetRef		17	20	17	18	18	21
GetArrivalpointNext		23	22	23	23	23	21
SetArrivalpointNext		19	14	19	22	22	19
TestArrivalpointWritable		30	34	30	30	30	32
GetExecutionTime		13	14	13	10	10	9
GetLargestExecutionTime		21	20	21	24	24	25
ResetLargestExecutionTime		10	11	10	16	16	15
GetStackOffset		41	36	42	42	39	44

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	120	154	181	125	149	211
	NS	107	144	172	106	141	190
	KL	58	88	125	68	99	147
TerminateTask	LExt	0	0	0	0	0	0
	H	538	544	530	548	539	552
ChainTask	SWL	725	761	854	819	822	922
	SWH	828	873	958	908	922	1030
	NSL	717	759	851	812	827	924
	NSH	814	866	949	909	920	1023
Schedule	SW	125	119	135	119	121	131
GetTaskID		34	33	37	32	36	39
GetTaskState		92	88	91	99	104	96
EnableAllInterrupts		34	38	36	36	33	35
DisableAllInterrupts		47	47	50	45	43	46
ResumeAllInterrupts		49	49	49	46	46	46
SuspendAllInterrupts		62	62	61	64	66	65
ResumeOSInterrupts		50	50	50	46	46	46
SuspendOSInterrupts		63	63	62	65	67	66
GetResource	Task	46	43	48	48	45	47
	Combined	90	90	87	83	90	87
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	101	102	101	107	105	105
	Combined	161	167	161	164	166	166
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	134	130	143
	NS	n/a	n/a	n/a	91	88	98
	KL	n/a	n/a	n/a	54	53	53
ClearEvent		n/a	n/a	n/a	66	68	66
GetEvent		n/a	n/a	n/a	27	26	27
WaitEvent	<default>	n/a	n/a	n/a	1028	1013	1053
	fp	n/a	n/a	n/a	1046	1017	1079
GetAlarmBase		83	87	80	80	75	78
GetAlarm		125	127	122	122	117	118

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		153	160	154	161	155	156
SetAbsAlarm		153	158	153	148	148	147
CancelAlarm		105	105	103	109	107	106
InitCounter		65	66	67	67	63	62
GetCounterValue		68	75	74	65	64	67
osek_tick_alarm	<default>	91	87	85	88	86	90
	KL	50	46	49	49	52	49
osek_incr_counter		15	15	15	14	14	14
GetActiveApplicationMode		8	8	8	8	8	8
StartOS		4367	4076	2986	4519	3448	2977
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	54	53	55	53	55	55
InitCOM		13	10	14	11	11	11
CloseCOM		10	13	11	15	11	11
StartCOM		62	55	66	154	158	157
StopCOM		24	24	24	20	22	22
ReadFlag		n/a	n/a	n/a	14	14	14
ResetFlag		n/a	n/a	n/a	9	9	9
ReceiveMessage		93	95	97	330	322	321
GetMessageResource		n/a	n/a	n/a	114	124	116
ReleaseMessageResource		n/a	n/a	n/a	173	174	174
GetMessageStatus		n/a	n/a	n/a	64	66	64
SendMessage	SW	243	277	302	482	492	552
	NS	224	261	287	459	498	545
	KL	124	150	195	371	398	440
ActivateTaskset	SW	106	686	763	113	706	781
	NS	96	706	783	101	508	854
	KL	47	627	764	58	648	723
	SW2	106	686	763	113	706	781
	NS2	96	706	783	101	508	854
	KL2	47	627	764	58	648	723
ChainTaskset	SWL	716	1170	1458	801	1386	1475
	SWH	815	1385	1545	907	1468	1643
	NSL	710	1138	1464	796	1349	1556
	NSH	819	1417	1365	899	1275	1543
GetTasksetRef		26	27	25	27	27	28

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		64	63	60	65	63	64
AssignTaskset		23	24	22	22	21	23
RemoveTaskset		61	63	59	61	62	63
TestSubTaskset		68	67	66	70	66	69
TestEquivalentTaskset		62	60	66	63	59	61
TickSchedule	SW	180	820	960	251	862	939
	NS	171	799	946	232	845	914
	KL	115	755	894	198	799	868
	SW2	183	820	960	251	846	929
	NS2	172	799	946	232	829	904
	KL2	114	753	892	197	782	857
AdvanceSchedule	SW	164	806	933	233	831	907
	NS	144	785	927	212	825	891
	KL	101	756	895	187	797	859
	SW2	164	806	933	233	815	898
	NS2	144	785	927	212	809	884
	KL2	102	757	896	186	780	849
StartSchedule		122	120	122	126	123	122
StopSchedule		109	108	109	108	110	108
GetScheduleStatus		121	125	121	120	123	121
GetScheduleValue		120	123	120	123	120	120
GetScheduleNext		29	30	29	25	26	26
SetScheduleNext		27	25	27	21	22	22
GetArrivalpointDelay		23	22	25	24	26	26
SetArrivalpointDelay		14	18	16	24	22	22
GetArrivalpointTasksetRef		24	20	20	18	19	19
GetArrivalpointNext		20	22	22	22	21	21
SetArrivalpointNext		16	14	19	23	23	23
TestArrivalpointWritable		32	34	30	28	28	28
GetExecutionTime		162	167	165	161	162	162
GetLargestExecutionTime		35	36	36	40	39	39
ResetLargestExecutionTime		23	24	24	28	29	29
GetStackOffset		42	42	37	45	40	41

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	434	494	502	460	472	498
	NS	456	522	540	493	517	530
	KL	363	420	432	388	408	423
TerminateTask	LExt	619	623	616	630	618	617
	H	711	705	699	698	692	700
ChainTask	SWL	1138	1199	1276	1241	1265	1329
	SWH	1234	1280	1353	1333	1344	1399
	NSL	1169	1229	1299	1277	1282	1335
	NSH	1273	1322	1396	1375	1370	1429
Schedule	SW	201	197	216	200	205	209
GetTaskID		43	41	42	41	42	45
GetTaskState		440	459	456	466	456	441
EnableAllInterrupts		44	46	46	50	46	45
DisableAllInterrupts		51	54	54	60	64	59
ResumeAllInterrupts		83	82	82	82	78	82
SuspendAllInterrupts		77	73	73	69	67	70
ResumeOSInterrupts		80	79	79	82	78	82
SuspendOSInterrupts		73	69	69	66	64	67
GetResource	Task	673	644	349	709	742	408
	Combined	349	349	358	413	416	392
	CLEx	360	361	367	421	435	412
ReleaseResource	Task	344	346	356	408	412	397
	Combined	368	371	378	432	435	417
	CLEx	336	329	342	389	397	386
SetEvent	SW	n/a	n/a	n/a	489	466	464
	NS	n/a	n/a	n/a	496	488	473
	KL	n/a	n/a	n/a	419	412	408
ClearEvent		n/a	n/a	n/a	135	131	133
GetEvent		n/a	n/a	n/a	394	386	380
WaitEvent	<default>	n/a	n/a	n/a	1155	1148	1165
	fp	n/a	n/a	n/a	1167	1152	1189
GetAlarmBase		288	301	320	300	302	314
GetAlarm		305	305	326	306	317	320

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		362	375	394	365	374	378
SetAbsAlarm		346	351	377	353	353	367
CancelAlarm		280	294	320	293	287	307
InitCounter		460	457	475	453	456	499
GetCounterValue		261	268	264	266	271	269
osek_tick_alarm	<default>	150	149	152	151	152	151
	KL	51	49	50	47	49	47
osek_incr_counter		15	15	15	15	15	15
GetActiveApplicationMode		5	5	5	8	8	8
StartOS		3567	3626	3076	4284	4678	4217
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	61	64	60	62	58	61
InitCOM		14	11	10	16	18	15
CloseCOM		15	14	13	13	14	10
StartCOM		89	87	84	179	180	173
StopCOM		40	39	38	40	42	40
ReadFlag		n/a	n/a	n/a	33	33	31
ResetFlag		n/a	n/a	n/a	35	34	31
ReceiveMessage		231	232	224	441	465	451
GetMessageResource		n/a	n/a	n/a	586	588	573
ReleaseMessageResource		n/a	n/a	n/a	587	593	568
GetMessageStatus		n/a	n/a	n/a	189	193	193
SendMessage	SW	683	745	758	936	961	974
	NS	704	772	795	965	1001	1002
	KL	562	621	633	805	840	840
ActivateTaskset	SW	771	1421	1432	782	1334	1463
	NS	799	1444	1473	851	1426	1435
	KL	645	1332	1406	710	1334	1370
	SW2	771	1421	1432	782	1334	1463
	NS2	799	1444	1473	851	1426	1435
	KL2	645	1332	1406	710	1334	1370
ChainTaskset	SWL	1522	2131	2262	1581	2157	2291
	SWH	1399	2227	2207	1692	2236	2443
	NSL	1402	2016	2279	1619	1993	2314
	NSH	1415	2226	2341	1689	2260	2378
GetTasksetRef		344	373	365	380	356	346

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		175	179	175	180	176	180
AssignTaskset		98	105	102	107	100	102
RemoveTaskset		180	186	179	184	178	176
TestSubTaskset		181	179	176	179	181	176
TestEquivalentTaskset		183	178	172	179	184	176
TickSchedule	SW	256	1578	1643	953	1594	1635
	NS	288	1610	1674	993	1632	1682
	KL	197	1524	1585	904	1543	1581
	SW2	256	1578	1638	953	1567	1611
	NS2	288	1610	1669	993	1604	1658
	KL2	198	1525	1581	903	1515	1556
AdvanceSchedule	SW	256	1572	1655	957	1605	1643
	NS	290	1592	1686	988	1631	1674
	KL	201	1517	1595	894	1547	1580
	SW2	256	1572	1650	957	1580	1619
	NS2	290	1592	1681	988	1606	1650
	KL2	198	1514	1587	895	1523	1557
StartSchedule		223	226	220	224	218	221
StopSchedule		167	171	168	177	177	173
GetScheduleStatus		190	188	186	187	183	189
GetScheduleValue		177	180	174	179	173	176
GetScheduleNext		73	76	72	77	73	74
SetScheduleNext		121	123	121	127	125	125
GetArrivalpointDelay		84	89	89	90	90	92
SetArrivalpointDelay		112	107	106	104	103	108
GetArrivalpointTasksetRef		77	76	73	78	75	83
GetArrivalpointNext		79	79	82	79	82	77
SetArrivalpointNext		117	126	119	123	116	119
TestArrivalpointWritable		80	82	88	83	89	79
GetExecutionTime		200	201	205	209	204	210
GetLargestExecutionTime		341	353	349	364	354	339
ResetLargestExecutionTime		321	343	328	338	329	315
GetStackOffset		38	38	39	39	42	39

4.3.2 OS Start-up Time

OS start-up time is the time from the entry to the `StartOS()` function to the execution of the first instruction in a user task (including the idle task) without any hook routines being called. This time is always application dependent, since `StartOS()` may activate any number of tasks and start any number of user-specified alarms.

4.3.3 Interrupt Latencies

Interrupt latency is the time between an interrupt request being recognized by the target hardware and the execution of the first instruction of the user provided handler function. The following tables give the interrupt latencies (in CPU cycles).

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	76	76	76	75	75	75
	Cat 2	132	177	177	180	177	180

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	76	76	76	75	75	75
	Cat 2	375	412	404	416	415	409

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	76	76	76	75	75	75
	Cat 2	378	407	411	409	413	405

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.

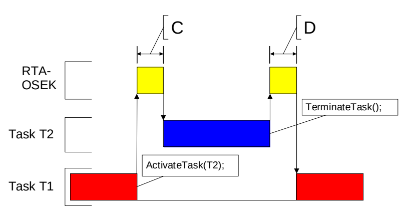

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

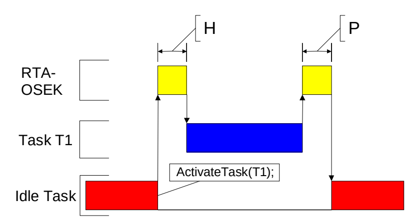

Figure 3: Task Activation from Idle Task

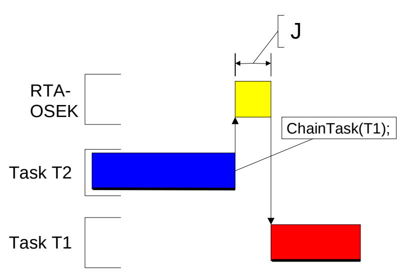

Figure 2: Task Chaining

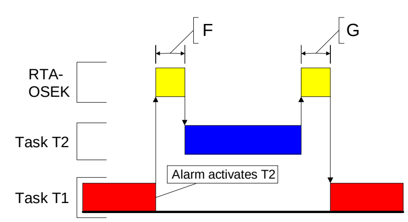

Figure 4: Task Activation from an Alarm

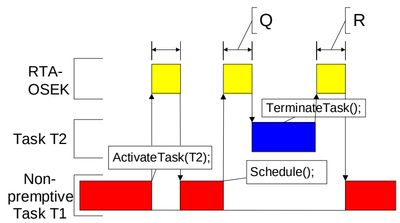

Figure 5: Non-Premptive Task Calls Schedule()

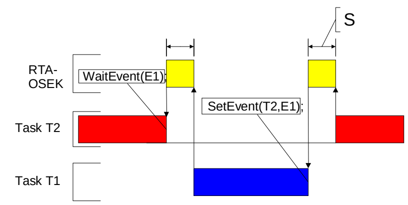

Figure 7: Waiting Task Activated by SetEvent()

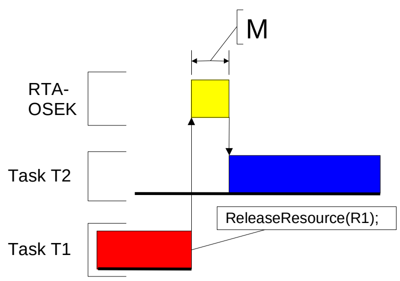

Figure 6: Blocked Task Activated by ReleaseResource()

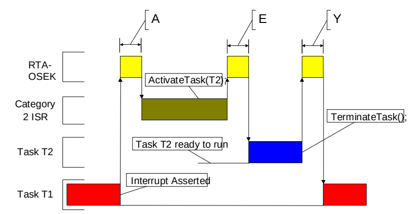

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	116	178	203	114	180	203
Figure 1: D	Heavy, Basic/Extended	239	296	324	307	308	331
ChainTask	Light, Basic	252	308	396	261	317	405
Figure 2: J	Heavy, Basic/Extended	663	783	884	739	798	921
Pre-emption	Light, Basic	193	251	338	199	254	365
Figure 1: C	Heavy, Basic/Extended	340	391	471	403	442	537
From idle task	Light, Basic	193	251	338	198	253	364
Figure 3: H	Heavy, Basic/Extended	340	391	471	402	441	536
Triggered by alarm	Light, Basic	309	362	450	314	372	475
Figure 4: F	Heavy, Basic/Extended	446	495	581	507	558	643
Schedule	Light, Basic	179	202	265	180	207	272
Figure 5: Q	Heavy, Basic/Extended	326	342	398	384	395	447
Release resource	Light, Basic	178	204	259	182	209	262
Figure 6: M	Heavy, Basic/Extended	325	344	392	386	397	437
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	686	694	844
From category 2 ISR	Light, Basic	202	276	324	244	277	324
Figure 8: E	Heavy, Basic/Extended	349	416	457	448	465	499

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	428	461	480	430	463	487
Figure 1: D	Heavy, Basic/Extended	540	563	577	586	578	610
ChainTask	Light, Basic	601	653	725	620	657	751
Figure 2: J	Heavy, Basic/Extended	1306	1387	1467	1363	1399	1536
Pre-emption	Light, Basic	394	433	512	395	440	553
Figure 1: C	Heavy, Basic/Extended	530	557	654	608	621	736
From idle task	Light, Basic	394	433	512	394	439	552
Figure 3: H	Heavy, Basic/Extended	530	557	654	607	620	735
Triggered by alarm	Light, Basic	512	547	619	511	552	668
Figure 4: F	Heavy, Basic/Extended	641	664	762	716	730	847
Schedule	Light, Basic	380	384	450	378	395	461
Figure 5: Q	Heavy, Basic/Extended	516	508	592	591	580	648
Release resource	Light, Basic	386	388	441	380	399	453
Figure 6: M	Heavy, Basic/Extended	522	512	583	593	584	640
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	898	880	1008
From category 2 ISR	Light, Basic	718	761	803	750	758	812
Figure 8: E	Heavy, Basic/Extended	854	880	943	961	941	997

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	601	644	659	611	640	656
Figure 1: D	Heavy, Basic/Extended	705	718	745	742	738	765
ChainTask	Light, Basic	1012	1088	1145	1039	1089	1158
Figure 2: J	Heavy, Basic/Extended	1881	1947	2035	1937	1969	2056
Pre-emption	Light, Basic	679	753	805	697	718	813
Figure 1: C	Heavy, Basic/Extended	815	877	945	912	923	994
From idle task	Light, Basic	681	755	807	700	721	816
Figure 3: H	Heavy, Basic/Extended	817	879	947	915	926	997
Triggered by alarm	Light, Basic	855	929	983	878	902	991
Figure 4: F	Heavy, Basic/Extended	989	1052	1121	1093	1101	1174
Schedule	Light, Basic	434	445	504	422	445	514
Figure 5: Q	Heavy, Basic/Extended	570	569	646	637	642	700
Release resource	Light, Basic	586	608	648	650	664	704
Figure 6: M	Heavy, Basic/Extended	722	732	790	865	861	890
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1237	1215	1310
From category 2 ISR	Light, Basic	758	808	842	785	799	856
Figure 8: E	Heavy, Basic/Extended	892	926	981	1000	993	1041

4.4 Configuration of Run-time Context

The run-time contexts of all tasks reside on the same stack and are recovered when the task terminates. As a result, run-time contexts of mutually exclusive tasks are effectively overlaid. The RTA-OSEK GUI is able to calculate the worst-case stack requirement for the entire application, based on the declared stack usage, the priorities and the resource occupation of individual tasks.

The size of the run-time context of a task depends on the task type and the system configuration. The following tables give the sizes (in bytes) for different OS status and configurations:

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		128	128	128	128	128	128
BCC1 lightweight, floating-point		144	144	144	144	144	144
BCC1 heavyweight, integer		240	240	240	240	240	240
BCC1 heavyweight, floating-point		240	240	240	240	240	240
BCC2 lightweight, integer		n/a	144	144	n/a	144	144
BCC2 lightweight, floating-point		n/a	144	144	n/a	144	144
BCC2 heavyweight, integer		n/a	256	256	n/a	256	256
BCC2 heavyweight, floating-point		n/a	256	256	n/a	256	256
ECC1 heavyweight, integer		n/a	n/a	n/a	288	288	288
ECC1 heavyweight, floating-point		n/a	n/a	n/a	288	288	288
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	256
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	256
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		128	128	144	128	128	144
BCC1 lightweight, floating-point		144	144	160	144	144	160
BCC1 heavyweight, integer		240	240	256	240	240	256
BCC1 heavyweight, floating-point		240	240	256	240	240	256
BCC2 lightweight, integer		n/a	144	160	n/a	144	160
BCC2 lightweight, floating-point		n/a	144	160	n/a	144	160
BCC2 heavyweight, integer		n/a	256	272	n/a	256	272
BCC2 heavyweight, floating-point		n/a	256	272	n/a	256	272
ECC1 heavyweight, integer		n/a	n/a	n/a	288	288	304
ECC1 heavyweight, floating-point		n/a	n/a	n/a	288	288	304
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	272
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	272

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		160	160	160	160	160	160
BCC1 lightweight, floating-point		176	176	176	176	176	176
BCC1 heavyweight, integer		288	288	288	288	288	288
BCC1 heavyweight, floating-point		288	288	288	288	288	288
BCC2 lightweight, integer		n/a	176	176	n/a	176	176
BCC2 lightweight, floating-point		n/a	176	176	n/a	176	176
BCC2 heavyweight, integer		n/a	288	288	n/a	288	288
BCC2 heavyweight, floating-point		n/a	288	288	n/a	288	288
ECC1 heavyweight, integer		n/a	n/a	n/a	320	320	320
ECC1 heavyweight, floating-point		n/a	n/a	n/a	320	320	320
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	288
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	288
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		160	160	176	160	160	176
BCC1 lightweight, floating-point		176	176	192	176	176	192
BCC1 heavyweight, integer		288	288	304	288	288	304
BCC1 heavyweight, floating-point		288	288	304	288	288	304
BCC2 lightweight, integer		n/a	176	192	n/a	176	192
BCC2 lightweight, floating-point		n/a	176	192	n/a	176	192
BCC2 heavyweight, integer		n/a	288	304	n/a	288	304
BCC2 heavyweight, floating-point		n/a	288	304	n/a	288	304
ECC1 heavyweight, integer		n/a	n/a	n/a	320	320	336
ECC1 heavyweight, floating-point		n/a	n/a	n/a	320	320	336
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	304
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	304

Extended

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Pre- and Post-Task hooks not used						
Task type						
BCC1 lightweight, integer	160	160	160	160	160	160
BCC1 lightweight, floating-point	176	176	176	176	176	176
BCC1 heavyweight, integer	288	288	288	288	288	288
BCC1 heavyweight, floating-point	288	288	288	288	288	288
BCC2 lightweight, integer	n/a	176	176	n/a	176	176
BCC2 lightweight, floating-point	n/a	176	176	n/a	176	176
BCC2 heavyweight, integer	n/a	288	288	n/a	288	288
BCC2 heavyweight, floating-point	n/a	288	288	n/a	288	288
ECC1 heavyweight, integer	n/a	n/a	n/a	320	320	320
ECC1 heavyweight, floating-point	n/a	n/a	n/a	320	320	320
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	288
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	288
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	160	160	176	160	160	176
BCC1 lightweight, floating-point	176	176	192	176	176	192
BCC1 heavyweight, integer	288	288	304	288	288	304
BCC1 heavyweight, floating-point	288	288	304	288	288	304
BCC2 lightweight, integer	n/a	176	192	n/a	176	192
BCC2 lightweight, floating-point	n/a	176	192	n/a	176	192
BCC2 heavyweight, integer	n/a	288	304	n/a	288	304
BCC2 heavyweight, floating-point	n/a	288	304	n/a	288	304
ECC1 heavyweight, integer	n/a	n/a	n/a	320	320	336
ECC1 heavyweight, floating-point	n/a	n/a	n/a	320	320	336
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	304
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	304

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for the MPC55XX/GreenHills 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. The inline versions of these API calls 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 5.04 Additions

6.1 Variants

The supported targets cores in the RTA-OSEK GUI are the e200z1 (MPC5514, MPC5516), e200z3 (MPC5534, SPC563M), e200z4 (MPC5643L, MPC5644A, MPC5645B, MPC5645C, MPC5646B, MPC5646C, SPC564A), e200z6 (MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, MPC5567), e200z7 (MPC5674F, MPC5675K) and generic MPC55xx

6.2 Kernel Linking

If an attempt to link a v5.0.4 OIL file with a v5.03 kernel library is made, a linker error will be seen. This is due to incompatible difference between v5.03 and v5.04 kernels.

7 Version 5.03 Additions

7.1 Variants

The supported targets cores in the RTA-OSEK GUI have been extended from the e200z1 (MPC5514, MPC5516), e200z3 (MPC5534), e200z6 (MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, MPC5567) and generic MPC55xx to additionally include the e200z3 (SPC563M), e200z4 (MPC5643L, MPC5644A, MPC5645B, MPC5645C, MPC5646B, MPC5646C, SPC564A) and e200z7 (MPC5674F, MPC5675K).

7.2 Floating Point Wrappers

EHI 101013: RTA-OSEK floating point save/restore model has been addressed.

7.3 Kernel Linking

If an attempt to link a v5.0.3 OIL file with a v5.02 kernel library is made, a linker error will be seen. This is due to incompatible difference between v5.02 and v5.03 kernels.

8 Version 5.02 Additions

The section names used by the kernel have been rationalized. The requirements for section initialization (zeroed, initialized, or initialised with zeroes), and the point this is needed (during C-startup or during StartOS) have been made clearer. The counters used by the interrupt manipulation APIs (and the inline variants) have been placed in the section os_cntr.

EHI call 81233 (relating to advanced counters) has been addressed.

EHI calls 85343 and 85350 relating to incorrect vector addresses have been addressed.

EHI calls 94016, 94017, and 96600 (relating to consequences of the race condition described in section 10.1) have been addressed.

9 Version 5.0.1 Additions

9.1 Variants

The supported targets in the RTA-OSEK GUI have been extended from the MPC5534, MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, MPC5567 and generic MPC55xx to additionally include the MPC5514 (Z1 core only) and MPC5516 (Z1 core only).

9.2 IVOR0 CPU Interrupt

In previous MPC55xx variants, for example, the MPC5534 the vector location IVOR 0 was defined as "Reserved". However, in more recent MPC55xx variants IVOR 0 has been redefined as "Critical Input" and in accordance with these developments it is now supported by version 5.0.1 of RTA-OSEK. It should be noted that IVOR 0 will require initializing in any user application because previous versions of RTA-OSEK only initialized from IVOR 1 onwards.

9.3 Linker Files

Linker command files (.ld) must be modified from the previous release to add the new memory sections (os_pur2, os_pir2 and os_pnir2).

10 Version 5 Changes

10.1 INTC Race Condition

A race condition between the modification of the INTC_CPR value and a triggering interrupt has been observed. For the race condition to occur the interrupt must trigger on the exact instruction that updates the value on the INTC_CPR. The result is that an implicit scheduling point is missed by RTA-OSEK. If tasks should be activated as a result of the interrupt in question the activation will occur at the next implicit scheduling point rather than at the end of the interrupt. If an application configures the INTC in software vector mode then the handled function must be modified along the guidelines section 3.1.8.

11 Compatibility with Pre-v5 Kernels

11.1 Updating the application version

To convert an existing v3.x OIL configuration file to v5.0.0, load the file into the v5 RTA-OSEK GUI, select the 'OS Configuration' option in the 'Application' menu and change the 'Kernel Version' to v5.00. When the OIL configuration file is saved it will then use the v5.0.0 format and the v5.00 kernel libraries. This process can be reversed to move back to earlier kernel versions.

11.2 Supporting software vector mode

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

Support

For product support, please contact your local ETAS representative.

Office locations and contact details can be found at the front of this manual and on the ETAS Group website www.etasgroup.com.