RTA-OSEK

Binding Manual: MPC55XX/Diab

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

Disclaimer

Trademarks

Contents

1 About this Guide

This guide provides target-specific information for the MPC55XX/Diab 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/Diab supports the single flat memory model supported by the Wind River Systems, Inc. (Diab) toolchain. This toolchain supports the Embedded Application Binary Interface, EABI.

2.1 Compiler

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

Vendor	Wind River Systems, Inc.
Compiler	Wind River (Diab) Compiler for PowerPC
Version	5.8.0.0-2_wind00198363

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

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

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

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

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

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

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

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

Option	Description
`-g`	Debugging must be disabled.

To support the use of multiple CPU configurations the environment variable CPU_TYPE should be set up to match the desired CPU target (e.g. PPC5554ES:simple).

2.2 Assembler

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

Vendor	Wind River Systems, Inc.
Assembler	Wind River (Diab) Assembler for the PowerPC
Version	5.8.0.0-2_wind00198363

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

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

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

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

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

2.3 Linker/Locator

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

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

2.4 Debugger

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

ORTI compatible debuggers	Lauterbach TRACE32

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

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of RTA-OSEK's interrupt model for MPC55XX/Diab. 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 Offset	Legality
0x0 to 0x22	The CPU vectors only handle Category 1 ISRs
0x10000 to the maximum INTC vector supported in 0x10 steps or 0x4 steps for z0/z1 core variants	The INTC vectors can handle Category 1 and 2 ISRs

The valid base addresses for the vector table are:

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

3.1.3 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Wind River Systems, Inc. C compiler can generate appropriate interrupt handling code for a C function decorated with the `__interrupt__` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

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

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

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

3.1.5 Vector Table Issues

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

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

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

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

3.1.6 Processor Mode

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

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

3.1.7 INTC vector mode

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

Provide the Interrupt handler for the SW mode:

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

```
interrupt_exception_handler:

; Code to save SRR0 and SRR1

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

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

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

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

```
os_mid_wrapper:
  stw   r4,44(r1)         ; Save interrupt context
                          ; following the EABI

                          ; Increment the interrupt
                          ; counter before enabling
                          ; global interrupts

  lwz   r4,os_isr_count@sdarx(r0)
  addi  r4,r4,1
  stw   r4,os_isr_count@sdarx(r0)

  mfspr r4,srr0           ; Save the SRR0/1 to
  stw   r4,12(r1)         ; allow nested interrupts
  mfspr r4,srr1
  stw   r4,16(r1)

  mtmsr r4                ; Restore pre-interrupted
                          ; msr (i.e. set EE bit and
                          ; SPE bit if enabled)

                          ; Cat 1 blocking ends here
  stw   r0,8(r1)

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

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

  stw   r5,48(r1)
  stw   r6,52(r1)
  stw   r7,56(r1)
  stw   r8,60(r1)
  stw   r9,64(r1)
  stw   r10,68(r1)
  stw   r11,72(r1)
  stw   r12,76(r1)

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

  bl    os_wrapper        ; Call inner wrapper

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

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

  lwz   r0,28(r1)         ; Restore the CRF with
                        ; a load inserted
  lwz   r5,48(r1)
  mtcrf 0xff,r0
  lwz   r0,24(r1)
  mtspr xer,r0
  lwz   r0,20(r1)
  mtspr ctr,r0
  lwz   r0,8(r1)

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

                        ; Return the IPL level to
                        ; that before the interrupt
                        ; triggered
                        ; INTC_CPR is at address
                        ; 0xFFF48008
  addis r4,r0,%hiadj(0xFFF48008)
  stw   r3,%lo(0xFFF48008)(r4)

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

```
 addi  r4,r4,-1
 stw   r4,os_isr_count@sdarx(r0)

 lwz   r4,16(r1)        ; Restore SRR0/1
 mtspr srr1,r4
 lwz   r4,12(r1)
 mtspr srr0,r4

 lwz   r4,44(r1)        ; Saved at the start of
                        ; os_mid_wrapper
 lwz   r3,40(r1)        ; Restore the remaining
                        ; context

 addi  r1,r1,80         ; Restore the SP

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

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

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

Generating a vector table and initializing the INTC_IACKR register

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

Initializing the IVOR4 registers:

The SW vector mode common interrupt exception handler's location is determined by an address derived from special purpose registers IVPR and IVOR4.

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

3.1.8 INTC and CPU Vector Offset Mapping

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

CPU interrupts and exceptions (Traditional Variants)

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

CPU interrupts and exceptions (z0 and z1 Core Variants)

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

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

```
.alignn 16
b   Floating_Point                    ; IVOR7
.alignn 16
b   System_Call                       ; IVOR8
.alignn 16
b   Not_Used                          ; IVOR9
.alignn 16
b   Decrementer                       ; IVOR10
.alignn 16
b   Fixed_Interval_Timer              ; IVOR11
.alignn 16
b   Watchdog_Timer                    ; IVOR12
.alignn 16
b   Data_TLB_Error                    ; IVOR13
.alignn 16
b   Instruction_TLB_Error             ; IVOR14
.alignn 16
b   Debug                             ; IVOR15
```

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

INTC interrupts and exceptions (Traditional Variants)

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

INTC interrupts and exceptions (z0 and z1 Core Variants)

INTC exception sources are characterized in an integer array `os_INTC_vectors`, as for traditional variants but instead of being 16-byte aligned interrupt exception handlers, they're 4-byte aligned. The array still requires aligning to a 64 Kbytes boundary and using the IVPR register to dictate the base address of the vector table. The table for the INTC vectors must be placed 2 Kbytes above the CPU vectors, an example of which is shown below.

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

        ...
}
```

3.1.9 Number of supported INTC vectors

The number of vectors available depends upon the PowerPC chip variant selected in RTA-OSEK. Currently the cores directly supported are the e200z1 (MPC5514, MPC5516), e200z3 (MPC5533, MPC5534, SPC563M), e200z4 (MPC5643L, MPC5644A, SPC564A), e200z6 (MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, MPC5567), e200z7 (MPC5674F) and the MPC55xx Generic. Further variants can be supported by contacting ETAS.

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

3.1.10 INTC PSR register initialization

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

3.1.11 INTC Race condition

A race condition between the modification of the INTC_CPR value and a triggering interrupt has been observed. For the race condition to occur the interrupt must trigger on the exact instruction that updates the value on the INTC_CPR. The result is that an implicit scheduling point is missed by RTA-

OSEK. So if a task should be activated as a result of the interrupt involved in this race condition, then the task activation will occur at the next implicit scheduling point rather than before the interrupted task is resumed. This problem has been resolved in v5.0.0 by the addition of an interrupt nesting counter `os_isr_count`. The counter can be addressed using the Small Data Area.

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

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

3.1.12 OS_LIFO_LOAD

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

3.1.13 Default Interrupt

The 'default interrupt' is intended to be used to catch all unexpected interrupts. All unused interrupts have their interrupt vectors directed to the named routine that you specify. This routine must correctly handle the interrupt context, in the same way as a Category 1 ISR.

Because RTA-OSEK only emits interrupt vectors for addresses 0x10000 up to the highest declared interrupt, it will only fill unused vectors with the default interrupt <u>up to the highest declared interrupt</u>. 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()`.

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

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

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

3.3 Stack Usage

3.3.1 Number of Stacks

A single stack is used. The first argument to `StackFaultHook` is always 0.

`osStackOffsetType` is a scalar, representing the number of bytes on the stack, with C type `unsigned long`.

3.3.2 Stack Usage within API Calls

The maximum stack usage within RTA-OSEK API calls, excluding calls to hooks and callbacks, is as follows:

Standard

API max usage (bytes): 128

Timing

API max usage (bytes): 128

Extended

API max usage (bytes): 176

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

3.3.3 Stack discipline

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

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

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

```
    os_SP_INIT = __SP_INIT;
```

3.4 Floating point

The Freescale PowerPC 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:

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

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

An example of how to save this floating-point context can be found in `osfptgt.c` and `osfptgt.h` in the <RTA-OSEK location>\diab55xx\inc directory. For an application to use floating-point context saving, the appropriate tasks and Category 2 interrupt service routines must be marked as using floating-point operation in the RTA-OSEK GUI. Note that the performance figures have been collected for a system not using hardware floating point.

4 Parameters of Implementation

This chapter provides detailed information on the functionality, performance and memory demands of the RTA-OSEK Component.

The RTA-OSEK Component is highly scalable. As a result, different figures will be obtained when your application uses different sets of features. These feature-sets give six classes of RTA-OSEK, depending on whether your application uses events, shared task priorities and/or multiple (queued) task activations. You should identify which class your application belongs to and then use the figures from the appropriate column in the table.

The following hardware was used to take the measurements in this chapter:

Processor	**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	64	64	64	64	64	64
Maximum number of not suspended tasks	64	64	64	64	64	64
Maximum number of priorities	64	64	64	64	64	64
Number of tasks per priority (for BCC2 and ECC2)	n/a	64	64	n/a	64	64
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	58	58	58	58	58	58
	ROM	194	194	198	286	286	290
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	78	78	78	78	78	78
	ROM	266	266	270	358	358	362
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	104	104	104	104	104	104
	ROM	332	332	336	424	424	428
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	44	44	44	44	44	44
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	48	48	48	48	48	48
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	56	64	n/a	56	64
ECC1, Integer task	RAM	n/a	n/a	n/a	116	116	116
	ROM	n/a	n/a	n/a	68	68	68
ECC1, floating-point task	RAM	n/a	n/a	n/a	120	120	120
	ROM	n/a	n/a	n/a	68	68	68
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	76
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	76
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	64	64	64	64	64	64
Category 2 ISR, floating-point	RAM	4	4	4	4	4	4
	ROM	100	100	100	100	100	100
Resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
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	24	24	24	24	24	24
Alarm	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
Counter	RAM	4	4	4	4	4	4
	ROM	128	128	128	128	128	128
Message	RAM	11	11	11	51	51	51
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
ScheduleTable	RAM	16	16	16	16	16	16
	ROM	128	128	128	128	128	128
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	16	16	16	16	16	16
Arrivalpoint (writable)	RAM	16	16	16	16	16	16
	ROM	16	16	16	16	16	16
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	8	8	8	8	8	8
Taskset (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	56	56	56	56	56	56
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	68	76	n/a	68	76
ECC1, Integer task	RAM	n/a	n/a	n/a	128	128	128
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	132	132	132
	ROM	n/a	n/a	n/a	80	80	80
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	88
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	88
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	112	112	112	112	112	112
Category 2 ISR, floating-point	RAM	16	16	16	16	16	16
	ROM	144	144	144	144	144	144
Resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
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	24	24	24	24	24	24
Alarm	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
Counter	RAM	4	4	4	4	4	4
	ROM	128	128	128	128	128	128
Message	RAM	11	11	11	51	51	51
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
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	128	128	128	128	128	128
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	16	16	16	16	16	16
Arrivalpoint (writable)	RAM	16	16	16	16	16	16
	ROM	16	16	16	16	16	16
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	8	8	8	8	8	8
Taskset (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8

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	68	68	68	68	68	68
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	68	68	68	68	68	68
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	76	84	n/a	76	84
ECC1, Integer task	RAM	n/a	n/a	n/a	132	132	132
	ROM	n/a	n/a	n/a	88	88	88
ECC1, floating-point task	RAM	n/a	n/a	n/a	136	136	136
	ROM	n/a	n/a	n/a	88	88	88
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	96
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	96
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	124	124	124	124	124	124
Category 2 ISR, floating-point	RAM	20	20	20	20	20	20
	ROM	156	156	156	156	156	156
Resource	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36

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	52	52	52	52	52	52
Counter	RAM	4	4	4	4	4	4
	ROM	132	132	132	132	132	132
Message	RAM	11	11	11	51	51	51
	ROM	24	24	24	60	60	60
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
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	128	128	128	128	128	128
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	24	24	24	24	24	24
Arrivalpoint (writable)	RAM	24	24	24	24	24	24
	ROM	24	24	24	24	24	24
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	8	8	8	8	8	8
Taskset (writable)	RAM	8	8	8	8	8	8
	ROM	8	8	8	8	8	8

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	244	376	440	268	392	504
	NS		208	340	404	224	356	468
	KL	2	144	268	332	160	300	412
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	232	364	416	248	396	496
	SWH	1, 9	252	412	464	276	428	520
	NSL	8	232	364	416	248	396	496
	NSH	9	240	400	452	264	416	508
Schedule			176	176	220	176	176	220
GetTaskID			36	36	36	36	36	36
GetTaskState			136	136	136	180	180	180
EnableAllInterrupts			40	40	40	40	40	40
DisableAllInterrupts			52	52	52	52	52	52
ResumeAllInterrupts			56	56	56	56	56	56
SuspendAllInterrupts			76	76	76	76	76	76
ResumeOSInterrupts			48	48	48	48	48	48
SuspendOSInterrupts			76	76	76	76	76	76
GetResource	Task	7	120	120	140	120	120	140
	Combined	6	204	204	204	204	204	204
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	136	136	136	136	136	136
	Combined	6	300	300	300	300	300	300
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	232	232	356
	NS		n/a	n/a	n/a	172	172	304
	NS1i	10	n/a	n/a	n/a	68	n/a	n/a
	KL	2	n/a	n/a	n/a	128	128	284
	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	328	328	604
	fp	11	n/a	n/a	n/a	372	372	692
	1i	10	n/a	n/a	n/a	24	n/a	n/a
GetAlarmBase			72	72	72	72	72	72
GetAlarm			164	164	164	164	164	164
SetRelAlarm			728	728	728	728	728	728
SetAbsAlarm			776	776	776	776	776	776
CancelAlarm			152	152	152	152	152	152
InitCounter			68	68	68	68	68	68
GetCounterValue			96	96	96	96	96	96
GetScheduleTableStatus		34	104	128	128	104	128	128
NextScheduleTable		34	156	428	428	156	428	428
StartScheduleTable		34	220	304	304	220	304	304
StopScheduleTable		34	144	200	200	144	200	200
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	64	64	64	64	64	64
Process container	Non-Yielding	33	36	36	36	36	36	36
osek_tick_alarm	<default>		104	104	104	104	104	104
	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	40	40	40	40	40	40
	Hook	13	52	52	52	52	52	52
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			60	60	60	60	60	60
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	112	112	112	300	300	300

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CCCB	15	300	300	300	300	300	300
GetMessageResource			88	88	88	88	88	88
ReleaseMessageResource			80	80	80	80	80	80
GetMessageStatus			76	76	76	76	76	76
SendMessage	SW CCCA	1, 14	152	152	152	372	372	372
	SW CCCB	1, 15	348	348	348	372	372	372
	NS CCCA	14	152	152	152	372	372	372
	NS CCCB	15	348	348	348	372	372	372
	KL CCCA	2, 14	116	116	116	328	328	328
	KL CCCB	2, 15	304	304	304	328	328	328
main_dispatch	NoHook	12	260	260	336	260	260	336
	Hook	13	300	300	376	300	300	376
sub_dispatch	B1LF	19	48	48	48	48	48	48
	B1HI	20	148	148	148	148	148	148
	B1HF	21	156	156	156	156	156	156
	B2LI	22	n/a	124	164	n/a	124	164
	B2LF	23	n/a	128	168	n/a	128	168
	B2HI	24	n/a	356	428	n/a	356	428
	B2HF	25	n/a	364	436	n/a	364	436
	E1HI	26	n/a	n/a	n/a	524	524	612
	E1HF	27	n/a	n/a	n/a	532	532	620
	E2HI	28	n/a	n/a	n/a	n/a	n/a	612
	E2HF	29	n/a	n/a	n/a	n/a	n/a	620
ErrorHook support		16	76	76	76	76	76	76
	ServiceID	17	84	84	84	84	84	84
	Parameters	18	100	100	100	100	100	100
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	236	460	532	268	512	624
	NS		200	424	496	232	476	588
	KL	2	136	400	472	168	452	540
ChainTaskset	SWL	1, 8	232	480	544	256	516	620
	SWH	1, 9	292	548	612	316	584	688
	NSL	8	232	480	544	256	516	620
	NSH	9	280	536	600	304	572	676
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			76	76	76	76	76	76
AssignTaskset			60	60	60	60	60	60
RemoveTaskset			76	76	76	76	76	76
TestSubTaskset			100	100	100	100	100	100
TestEquivalentTaskset			108	108	108	108	108	108
TickSchedule	SW	1	364	264	264	264	264	264
	NS		320	244	244	244	244	244
	KL	2	272	196	196	196	196	196
AdvanceSchedule	SW	1	320	236	236	236	236	236
	NS		276	192	192	192	192	192
	KL	2	252	184	184	184	184	184
StartSchedule			156	156	156	156	156	156
StopSchedule			128	128	128	128	128	128
GetScheduleStatus			180	180	180	180	180	180
GetScheduleValue			148	148	148	148	148	148
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			56	56	56	56	56	56
GetExecutionTime			8	8	8	8	8	8
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			8	8	8	8	8	8
GetStackOffset			28	28	28	28	28	28

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	244	376	440	268	392	504
	NS		208	340	404	224	356	468
	KL	2	144	268	332	160	300	412
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	232	364	416	248	396	496
	SWH	1, 9	252	412	464	276	428	520
	NSL	8	232	364	416	248	396	496
	NSH	9	240	400	452	264	416	508
Schedule			204	204	248	204	204	248
GetTaskID			36	36	36	36	36	36
GetTaskState			136	136	136	180	180	180
EnableAllInterrupts			40	40	40	40	40	40
DisableAllInterrupts			52	52	52	52	52	52
ResumeAllInterrupts			56	56	56	56	56	56
SuspendAllInterrupts			76	76	76	76	76	76
ResumeOSInterrupts			48	48	48	48	48	48
SuspendOSInterrupts			76	76	76	76	76	76
GetResource	Task	7	120	120	140	120	120	140
	Combined	6	204	204	204	204	204	204
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	164	164	164	164	164	164
	Combined	6	352	352	352	352	352	352
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	232	232	356
	NS		n/a	n/a	n/a	172	172	304
	NS1i	10	n/a	n/a	n/a	68	n/a	n/a
	KL	2	n/a	n/a	n/a	128	128	284
	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	400	400	680
	fp	11	n/a	n/a	n/a	444	444	768

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	1i	10	n/a	n/a	n/a	136	n/a	n/a
GetAlarmBase			72	72	72	72	72	72
GetAlarm			164	164	164	164	164	164
SetRelAlarm			728	728	728	728	728	728
SetAbsAlarm			776	776	776	776	776	776
CancelAlarm			152	152	152	152	152	152
InitCounter			68	68	68	68	68	68
GetCounterValue			96	96	96	96	96	96
GetScheduleTableStatus		34	104	128	128	104	128	128
NextScheduleTable		34	156	428	428	156	428	428
StartScheduleTable		34	220	304	304	220	304	304
StopScheduleTable		34	144	200	200	144	200	200
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	64	64	64	64	64	64
Process container	Non-Yielding	33	36	36	36	36	36	36
osek_tick_alarm	<default>		104	104	104	104	104	104
	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			364	364	364	364	364	364
ShutdownOS	NoHook	12	40	40	40	40	40	40
	Hook	13	52	52	52	52	52	52
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			60	60	60	60	60	60
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	112	112	112	300	300	300
	CCCB	15	300	300	300	300	300	300
GetMessageResource			88	88	88	88	88	88
ReleaseMessageResource			80	80	80	80	80	80

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
GetMessageStatus			76	76	76	76	76	76
SendMessage	SW CCCA	1, 14	152	152	152	372	372	372
	SW CCCB	1, 15	348	348	348	372	372	372
	NS CCCA	14	152	152	152	372	372	372
	NS CCCB	15	348	348	348	372	372	372
	KL CCCA	2, 14	116	116	116	328	328	328
	KL CCCB	2, 15	304	304	304	328	328	328
main_dispatch	NoHook	12	324	324	400	324	324	400
	Hook	13	368	368	444	368	368	444
sub_dispatch	B1LF	19	36	36	36	36	36	36
	B1HI	20	144	144	144	144	144	144
	B1HF	21	152	152	152	152	152	152
	B2LI	22	n/a	96	140	n/a	96	140
	B2LF	23	n/a	100	144	n/a	100	144
	B2HI	24	n/a	332	408	n/a	332	408
	B2HF	25	n/a	340	416	n/a	340	416
	E1HI	26	n/a	n/a	n/a	580	580	668
	E1HF	27	n/a	n/a	n/a	588	588	676
	E2HI	28	n/a	n/a	n/a	n/a	n/a	668
	E2HF	29	n/a	n/a	n/a	n/a	n/a	676
ErrorHook support		16	76	76	76	76	76	76
	ServiceID	17	84	84	84	84	84	84
	Parameters	18	100	100	100	100	100	100
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	108	108	108	108	108	108
Timing_termination		4	132	132	132	132	132	132
ActivateTaskset	SW	1	236	460	532	268	512	624
	NS		200	424	496	232	476	588
	KL	2	136	400	472	168	452	540
ChainTaskset	SWL	1, 8	232	480	544	256	516	620
	SWH	1, 9	292	548	612	316	584	688
	NSL	8	232	480	544	256	516	620
	NSH	9	280	536	600	304	572	676
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			76	76	76	76	76	76
AssignTaskset			60	60	60	60	60	60
RemoveTaskset			76	76	76	76	76	76

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestSubTaskset			100	100	100	100	100	100
TestEquivalentTaskset			108	108	108	108	108	108
TickSchedule	SW	1	364	264	264	264	264	264
	NS		320	244	244	244	244	244
	KL	2	272	196	196	196	196	196
AdvanceSchedule	SW	1	320	236	236	236	236	236
	NS		276	192	192	192	192	192
	KL	2	252	184	184	184	184	184
StartSchedule			156	156	156	156	156	156
StopSchedule			128	128	128	128	128	128
GetScheduleStatus			180	180	180	180	180	180
GetScheduleValue			148	148	148	148	148	148
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			56	56	56	56	56	56
GetExecutionTime			156	156	156	156	156	156
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			20	20	20	20	20	20
GetStackOffset			28	28	28	28	28	28

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	400	528	560	416	520	612
	NS		520	648	704	536	640	732
	KL	2	288	408	472	304	440	516
TerminateTask	LExt	3	172	172	172	172	172	172

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	H	5	220	220	220	220	220	220
ChainTask	SWL	1, 8	452	592	644	468	592	692
	SWH	1, 9	500	604	652	516	620	704
	NSL	8	584	724	776	600	740	840
	NSH	9	636	740	788	652	756	840
Schedule			384	384	428	384	384	428
GetTaskID			60	60	60	60	60	60
GetTaskState			348	348	348	368	368	368
EnableAllInterrupts			64	64	64	64	64	64
DisableAllInterrupts			76	76	76	76	76	76
ResumeAllInterrupts			140	140	140	140	140	140
SuspendAllInterrupts			100	100	100	100	100	100
ResumeOSInterrupts			132	132	132	132	132	132
SuspendOSInterrupts			100	100	100	100	100	100
GetResource	Task	7	560	560	536	560	560	536
	Combined	6	572	572	572	572	572	572
	CLEx	3	488	488	488	488	488	488
ReleaseResource	Task	7	512	512	512	512	512	512
	Combined	6	700	700	700	700	700	700
	CLEx	3	488	488	488	488	488	488
SetEvent	SW	1	n/a	n/a	n/a	440	440	588
	NS		n/a	n/a	n/a	548	548	696
	NS1i	10	n/a	n/a	n/a	324	n/a	n/a
	KL	2	n/a	n/a	n/a	356	356	488
	KL1i	2, 10	n/a	n/a	n/a	276	n/a	n/a
ClearEvent			n/a	n/a	n/a	204	204	204
GetEvent			n/a	n/a	n/a	284	284	284
WaitEvent	<default>		n/a	n/a	n/a	544	544	808
	fp	11	n/a	n/a	n/a	588	588	896
	1i	10	n/a	n/a	n/a	316	n/a	n/a
GetAlarmBase			280	280	280	280	280	280
GetAlarm			252	252	252	252	252	252
SetRelAlarm			904	904	904	904	904	904
SetAbsAlarm			920	920	920	920	920	920
CancelAlarm			256	256	256	256	256	256
InitCounter			316	316	316	316	316	316
GetCounterValue			304	304	304	304	304	304

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleTableStatus		34	132	156	156	132	156	156
NextScheduleTable		34	180	452	452	180	452	452
StartScheduleTable		34	244	328	328	244	328	328
StopScheduleTable		34	168	208	208	168	208	208
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	76	76	76	76	76	76
Process container	Yielding	32	64	64	64	64	64	64
Process container	Non-Yielding	33	36	36	36	36	36	36
osek_tick_alarm	<default>		204	204	204	204	204	204
	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			388	388	388	388	388	388
ShutdownOS	NoHook	12	52	52	52	52	52	52
	Hook	13	64	64	64	64	64	64
InitCOM			8	8	8	8	8	8
CloseCOM			8	8	8	8	8	8
StartCOM			84	84	84	84	84	84
StopCOM			68	68	68	68	68	68
ReadFlag			40	40	40	40	40	40
ResetFlag			52	52	52	52	52	52
ReceiveMessage	CCCA	14	228	228	228	396	396	396
	CCCB	15	396	396	396	396	396	396
GetMessageResource			156	156	156	156	156	156
ReleaseMessageResource			156	156	156	156	156	156
GetMessageStatus			176	176	176	176	176	176
SendMessage	SW CCCA	1, 14	308	308	308	508	508	508
	SW CCCB	1, 15	484	484	484	508	508	508
	NS CCCA	14	308	308	308	508	508	508
	NS CCCB	15	484	484	484	508	508	508
	KL CCCA	2, 14	244	244	244	440	440	440
	KL CCCB	2, 15	416	416	416	440	440	440
main_dispatch	NoHook	12	324	324	400	324	324	400

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Hook	13	368	368	444	368	368	444
sub_dispatch	B1LF	19	36	36	36	36	36	36
	B1HI	20	144	144	144	144	144	144
	B1HF	21	152	152	152	152	152	152
	B2LI	22	n/a	96	140	n/a	96	140
	B2LF	23	n/a	100	144	n/a	100	144
	B2HI	24	n/a	332	408	n/a	332	408
	B2HF	25	n/a	340	416	n/a	340	416
	E1HI	26	n/a	n/a	n/a	580	580	668
	E1HF	27	n/a	n/a	n/a	588	588	676
	E2HI	28	n/a	n/a	n/a	n/a	n/a	668
	E2HF	29	n/a	n/a	n/a	n/a	n/a	676
ErrorHook support		16	180	180	180	180	180	180
	ServiceID	17	188	188	188	188	188	188
	Parameters	18	208	208	208	208	208	208
validity_checks		3	52	52	52	52	52	52
Timing_dispatch		4	108	108	108	108	108	108
Timing_termination		4	132	132	132	132	132	132
ActivateTaskset	SW	1	504	596	668	528	640	748
	NS		612	704	776	636	748	860
	KL	2	424	492	564	424	536	644
ChainTaskset	SWL	1, 8	580	676	740	596	712	816
	SWH	1, 9	656	760	820	672	796	896
	NSL	8	704	800	864	720	836	936
	NSH	9	768	872	936	784	904	1008
GetTasksetRef			224	224	224	224	224	224
MergeTaskset			380	380	380	380	380	380
AssignTaskset			372	372	372	372	372	372
RemoveTaskset			380	380	380	380	380	380
TestSubTaskset			404	404	404	404	404	404
TestEquivalentTaskset			396	396	396	396	396	396
TickSchedule	SW	1	512	412	412	412	412	412
	NS		624	548	548	548	548	548
	KL	2	444	336	336	336	336	336
AdvanceSchedule	SW	1	516	412	412	412	412	412
	NS		628	548	548	548	548	548
	KL	2	428	352	352	352	352	352

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StartSchedule			352	352	352	352	352	352
StopSchedule			284	284	284	284	284	284
GetScheduleStatus			348	348	348	348	348	348
GetScheduleValue			284	284	284	284	284	284
GetScheduleNext			136	136	136	136	136	136
SetScheduleNext			260	260	260	260	260	260
GetArrivalpointDelay			196	196	196	196	196	196
SetArrivalpointDelay			224	224	224	224	224	224
GetArrivalpointTasksetRef			192	192	192	192	192	192
GetArrivalpointNext			196	196	196	196	196	196
SetArrivalpointNext			280	280	280	280	280	280
TestArrivalpointWritable			228	228	228	228	228	228
GetExecutionTime			220	220	220	220	220	220
GetLargestExecutionTime			180	180	180	180	180	180
ResetLargestExecutionTime			164	164	164	164	164	164
GetStackOffset			28	28	28	28	28	28

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	167	218	268	181	228	288
	NS	146	203	264	155	210	282
	KL	93	144	200	109	166	233
TerminateTask	LExt	0	0	0	0	0	0
	H	264	272	268	270	270	270
ChainTask	SWL	442	515	610	519	570	694
	SWH	553	640	739	624	682	797
	NSL	444	516	614	515	571	701
	NSH	541	628	735	615	672	801
Schedule	SW	151	151	164	150	150	166
GetTaskID		39	39	34	33	36	31
GetTaskState		104	102	102	132	129	125
EnableAllInterrupts		45	45	45	50	48	48
DisableAllInterrupts		47	47	46	47	48	47
ResumeAllInterrupts		61	61	61	61	62	61
SuspendAllInterrupts		63	63	65	64	66	69
ResumeOSInterrupts		62	62	62	57	58	57
SuspendOSInterrupts		64	64	66	64	66	69
GetResource	Task	114	114	118	108	111	116
	Combined	126	123	126	128	129	126
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	116	116	114	115	113	110
	Combined	181	178	179	179	186	189
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	159	152	151
	NS	n/a	n/a	n/a	130	138	135
	KL	n/a	n/a	n/a	88	89	100
ClearEvent		n/a	n/a	n/a	74	72	76
GetEvent		n/a	n/a	n/a	30	32	30
WaitEvent	<default>	n/a	n/a	n/a	738	734	798
	fp	n/a	n/a	n/a	747	753	822
GetAlarmBase		91	86	89	87	88	91
GetAlarm		127	127	127	127	126	126

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		171	173	169	175	167	166
SetAbsAlarm		188	180	180	181	180	181
CancelAlarm		111	107	111	111	112	113
InitCounter		72	72	68	70	74	70
GetCounterValue		86	86	84	85	87	83
osek_tick_alarm	<default>	87	81	87	87	81	89
	KL	46	48	46	50	47	47
osek_incr_counter		21	21	21	20	20	20
GetActiveApplicationMode		8	8	8	11	11	11
StartOS		1396	1391	1338	1335	1537	1401
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	61	57	58	60	63	68
InitCOM		15	14	17	20	20	22
CloseCOM		22	20	21	17	18	18
StartCOM		72	74	71	119	111	120
StopCOM		27	29	29	27	25	27
ReadFlag		n/a	n/a	n/a	19	19	19
ResetFlag		n/a	n/a	n/a	15	15	15
ReceiveMessage		97	97	94	326	317	322
GetMessageResource		n/a	n/a	n/a	206	192	208
ReleaseMessageResource		n/a	n/a	n/a	189	199	199
GetMessageStatus		n/a	n/a	n/a	60	58	60
SendMessage	SW	287	333	380	522	575	630
	NS	267	322	387	517	551	633
	KL	172	228	279	418	459	540
ActivateTaskset	SW	127	1506	1680	139	1508	1719
	NS	112	1618	1667	125	1494	1702
	KL	66	1463	1765	72	1462	1663
	SW2	127	1506	1680	139	1508	1719
	NS2	112	1618	1667	125	1494	1702
	KL2	66	1463	1765	72	1462	1663
ChainTaskset	SWL	409	1936	2181	479	1970	2245
	SWH	542	1929	2166	602	1966	2238
	NSL	406	1801	2038	473	1970	1993
	NSH	531	1991	2169	600	2089	2234
GetTasksetRef		28	26	26	29	30	30

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		80	83	83	79	78	78
AssignTaskset		67	65	65	68	67	67
RemoveTaskset		70	70	70	80	77	72
TestSubTaskset		84	89	89	86	86	86
TestEquivalentTaskset		98	93	93	91	98	96
TickSchedule	SW	224	1667	1976	274	1679	1876
	NS	205	1647	1956	255	1657	1853
	KL	151	1592	1907	207	1612	1803
	SW2	224	1667	1976	274	1662	1867
	NS2	205	1647	1956	255	1640	1844
	KL2	151	1592	1907	207	1595	1794
AdvanceSchedule	SW	191	1633	1942	244	1655	1845
	NS	167	1611	1916	220	1628	1821
	KL	141	1582	1894	190	1592	1794
	SW2	191	1633	1942	244	1638	1836
	NS2	167	1611	1916	220	1611	1812
	KL2	141	1582	1894	190	1575	1785
StartSchedule		136	131	136	139	139	139
StopSchedule		123	119	118	119	119	124
GetScheduleStatus		135	143	134	136	132	132
GetScheduleValue		130	130	133	132	122	128
GetScheduleNext		30	30	31	30	31	31
SetScheduleNext		31	31	29	27	27	26
GetArrivalpointDelay		29	28	29	27	27	29
SetArrivalpointDelay		27	25	22	18	25	21
GetArrivalpointTasksetRef		17	20	24	24	18	20
GetArrivalpointNext		27	26	24	26	28	28
SetArrivalpointNext		26	22	23	22	25	25
TestArrivalpointWritable		37	37	36	36	37	34
GetExecutionTime		16	17	19	16	17	17
GetLargestExecutionTime		27	26	27	23	23	27
ResetLargestExecutionTime		22	20	19	21	21	20
GetStackOffset		28	27	26	26	29	27

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	171	221	263	170	220	288
	NS	136	214	247	143	195	260
	KL	90	151	191	101	157	221
TerminateTask	LExt	0	0	0	0	0	0
	H	623	626	622	622	627	627
ChainTask	SWL	859	921	1012	943	985	1091
	SWH	946	1042	1127	1040	1089	1180
	NSL	858	928	1018	944	987	1082
	NSH	938	1034	1117	1036	1086	1177
Schedule	SW	148	150	162	150	151	173
GetTaskID		39	38	38	33	37	38
GetTaskState		110	105	108	125	131	130
EnableAllInterrupts		45	48	47	50	51	51
DisableAllInterrupts		46	48	45	47	50	48
ResumeAllInterrupts		61	63	60	61	64	61
SuspendAllInterrupts		65	62	63	64	63	63
ResumeOSInterrupts		62	64	61	57	60	57
SuspendOSInterrupts		66	63	64	64	63	63
GetResource	Task	112	114	116	111	117	115
	Combined	126	127	123	125	129	126
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	115	114	114	116	115	116
	Combined	182	184	184	187	183	181
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	153	153	152
	NS	n/a	n/a	n/a	139	135	139
	KL	n/a	n/a	n/a	90	89	95
ClearEvent		n/a	n/a	n/a	72	76	73
GetEvent		n/a	n/a	n/a	33	30	31
WaitEvent	<default>	n/a	n/a	n/a	1112	1104	1166
	fp	n/a	n/a	n/a	1133	1116	1175
GetAlarmBase		84	89	87	88	88	91
GetAlarm		130	127	129	126	126	130

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
SetRelAlarm		169	169	173	167	170	167
SetAbsAlarm		177	180	182	180	180	174
CancelAlarm		109	111	107	112	109	109
InitCounter		72	70	70	74	74	74
GetCounterValue		86	82	82	81	85	87
osek_tick_alarm	<default>	83	87	81	81	83	83
	KL	47	46	48	47	49	48
osek_incr_counter		21	21	21	20	20	20
GetActiveApplicationMode		8	8	8	11	11	11
StartOS		3180	3342	3336	3072	3096	3338
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	61	59	59	67	61	59
InitCOM		14	19	19	19	22	21
CloseCOM		20	22	22	17	18	15
StartCOM		71	71	75	110	125	120
StopCOM		29	29	29	28	27	27
ReadFlag		n/a	n/a	n/a	19	19	19
ResetFlag		n/a	n/a	n/a	15	15	15
ReceiveMessage		97	98	98	332	319	323
GetMessageResource		n/a	n/a	n/a	200	200	199
ReleaseMessageResource		n/a	n/a	n/a	192	198	197
GetMessageStatus		n/a	n/a	n/a	60	60	60
SendMessage	SW	288	335	377	525	553	630
	NS	258	337	366	489	547	602
	KL	171	230	275	397	456	524
ActivateTaskset	SW	130	1509	1812	137	1511	1716
	NS	114	1553	1668	119	1491	1695
	KL	64	1525	1582	75	1458	1790
	SW2	130	1509	1812	137	1511	1716
	NS2	114	1553	1668	119	1491	1695
	KL2	64	1525	1582	75	1458	1790
ChainTaskset	SWL	815	2216	2384	909	2327	2502
	SWH	940	2332	2681	1027	2377	2493
	NSL	812	2217	2575	904	2327	2503
	NSH	932	2332	2548	1018	2431	2608
GetTasksetRef		28	26	27	30	29	29

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		80	87	78	79	80	76
AssignTaskset		68	68	65	68	68	65
RemoveTaskset		75	76	74	72	81	76
TestSubTaskset		84	87	89	86	86	91
TestEquivalentTaskset		100	96	94	96	91	92
TickSchedule	SW	231	1724	1777	273	1672	2008
	NS	205	1703	1756	253	1651	1985
	KL	152	1662	1715	203	1612	1935
	SW2	231	1724	1777	273	1655	1995
	NS2	205	1703	1756	253	1634	1972
	KL2	152	1662	1715	203	1595	1922
AdvanceSchedule	SW	196	1699	1750	250	1646	1975
	NS	171	1673	1728	223	1626	1954
	KL	143	1651	1699	187	1592	1914
	SW2	196	1699	1750	250	1629	1962
	NS2	171	1673	1728	223	1609	1941
	KL2	143	1651	1699	187	1575	1901
StartSchedule		136	139	139	139	142	141
StopSchedule		124	122	122	123	124	124
GetScheduleStatus		132	136	136	135	132	128
GetScheduleValue		129	127	127	129	126	128
GetScheduleNext		32	33	33	29	32	31
SetScheduleNext		28	28	28	29	26	26
GetArrivalpointDelay		29	29	29	28	27	26
SetArrivalpointDelay		27	22	22	24	18	23
GetArrivalpointTasksetRef		17	23	23	22	23	20
GetArrivalpointNext		27	26	26	27	27	27
SetArrivalpointNext		26	22	22	26	19	19
TestArrivalpointWritable		37	37	37	36	37	37
GetExecutionTime		201	198	198	200	198	200
GetLargestExecutionTime		41	43	43	42	39	39
ResetLargestExecutionTime		34	34	34	36	34	35
GetStackOffset		29	26	27	29	28	28

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	412	472	571	442	485	534
	NS	467	535	635	496	535	596
	KL	332	388	497	365	419	472
TerminateTask	LExt	681	678	676	671	672	681
	H	788	784	778	775	772	768
ChainTask	SWL	1180	1253	1425	1290	1327	1435
	SWH	1282	1350	1508	1380	1424	1519
	NSL	1254	1327	1500	1344	1396	1508
	NSH	1340	1405	1573	1437	1485	1594
Schedule	SW	229	233	250	218	222	238
GetTaskID		43	45	45	39	39	40
GetTaskState		407	413	469	442	443	435
EnableAllInterrupts		57	59	54	60	60	60
DisableAllInterrupts		64	61	69	65	65	65
ResumeAllInterrupts		95	88	90	85	85	85
SuspendAllInterrupts		76	82	74	76	76	76
ResumeOSInterrupts		92	85	87	87	87	87
SuspendOSInterrupts		72	78	70	76	76	76
GetResource	Task	798	832	405	859	855	431
	Combined	353	353	373	412	412	410
	CLEx	381	381	399	433	433	430
ReleaseResource	Task	378	369	394	416	416	416
	Combined	389	385	403	441	441	439
	CLEx	358	358	356	401	402	402
SetEvent	SW	n/a	n/a	n/a	456	453	459
	NS	n/a	n/a	n/a	490	488	487
	KL	n/a	n/a	n/a	404	400	403
ClearEvent		n/a	n/a	n/a	134	134	134
GetEvent		n/a	n/a	n/a	360	360	365
WaitEvent	<default>	n/a	n/a	n/a	1211	1212	1281
	fp	n/a	n/a	n/a	1233	1234	1289
GetAlarmBase		285	287	336	288	292	305
GetAlarm		298	287	334	302	295	308

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		353	352	407	360	366	382
SetAbsAlarm		356	354	401	369	363	382
CancelAlarm		270	276	325	281	286	310
InitCounter		490	477	536	477	475	531
GetCounterValue		267	275	298	274	272	277
osek_tick_alarm	<default>	157	148	148	153	148	148
	KL	45	45	46	46	45	44
osek_incr_counter		18	18	18	25	25	25
GetActiveApplicationMode		13	13	13	10	10	10
StartOS		3573	3698	3406	3426	3426	3169
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	63	63	68	70	70	67
InitCOM		20	20	20	14	14	17
CloseCOM		18	18	17	20	20	20
StartCOM		91	89	76	146	142	141
StopCOM		42	42	44	46	46	46
ReadFlag		n/a	n/a	n/a	42	42	39
ResetFlag		n/a	n/a	n/a	38	38	37
ReceiveMessage		240	239	236	455	451	444
GetMessageResource		n/a	n/a	n/a	613	613	603
ReleaseMessageResource		n/a	n/a	n/a	577	577	572
GetMessageStatus		n/a	n/a	n/a	174	174	172
SendMessage	SW	668	734	835	912	954	1011
	NS	724	797	907	962	1006	1063
	KL	536	592	712	789	841	892
ActivateTaskset	SW	1698	3086	3204	1708	3209	3164
	NS	1739	3194	3307	1753	3317	3084
	KL	1635	3016	3456	1515	3014	3422
	SW2	1698	3086	3204	1708	3209	3164
	NS2	1739	3194	3307	1753	3317	3084
	KL2	1635	3016	3456	1515	3014	3422
ChainTaskset	SWL	2493	3945	4117	2569	4044	4134
	SWH	2481	3879	4230	2668	4029	4351
	NSL	2540	4193	4185	2620	3982	4187
	NSH	2765	4167	4279	2723	4143	4271
GetTasksetRef		308	311	364	327	327	325

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		205	204	205	205	211	203
AssignTaskset		200	196	191	199	202	203
RemoveTaskset		202	199	202	202	200	208
TestSubTaskset		205	212	207	204	211	207
TestEquivalentTaskset		210	208	206	211	204	210
TickSchedule	SW	316	3273	3711	1765	3291	3698
	NS	363	3326	3758	1829	3355	3756
	KL	235	3194	3631	1706	3232	3622
	SW2	316	3273	3713	1771	3269	3680
	NS2	363	3326	3760	1830	3328	3740
	KL2	235	3194	3633	1708	3206	3603
AdvanceSchedule	SW	304	3256	3709	1748	3285	3684
	NS	352	3305	3753	1805	3325	3726
	KL	218	3186	3634	1686	3210	3613
	SW2	303	3255	3706	1748	3257	3664
	NS2	349	3302	3748	1805	3297	3706
	KL2	217	3185	3631	1686	3182	3593
StartSchedule		230	230	228	230	230	230
StopSchedule		181	169	172	176	176	171
GetScheduleStatus		204	197	197	192	192	197
GetScheduleValue		187	188	187	186	186	191
GetScheduleNext		68	70	77	70	70	71
SetScheduleNext		122	119	120	124	124	120
GetArrivalpointDelay		91	91	88	97	97	91
SetArrivalpointDelay		111	111	115	111	111	105
GetArrivalpointTasksetRef		80	80	81	81	81	75
GetArrivalpointNext		85	85	81	83	83	82
SetArrivalpointNext		129	129	124	125	125	127
TestArrivalpointWritable		96	96	90	94	94	92
GetExecutionTime		235	235	237	232	232	239
GetLargestExecutionTime		289	289	344	303	303	302
ResetLargestExecutionTime		274	274	336	297	297	291
GetStackOffset		24	25	27	28	27	28

4.3.2 OS Start-up Time

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

4.3.3 Interrupt Latencies

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

Standard

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
Operation	ISR Category						
ISR Latency	Cat 1	83	83	83	83	83	83
	Cat 2	132	219	219	214	214	222

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	83	83	83	83	83	83
	Cat 2	440	526	533	521	517	508

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	83	83	83	83	83	83
	Cat 2	441	521	511	517	525	504

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.

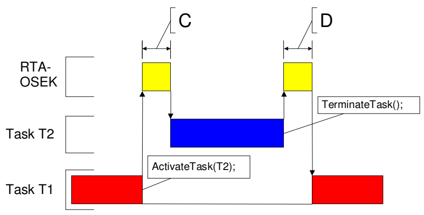

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

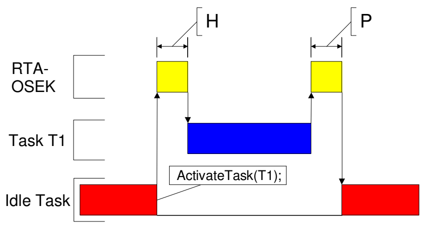

Figure 3: Task Activation from Idle Task

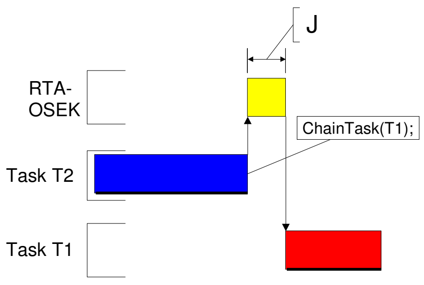

Figure 2: Task Chaining

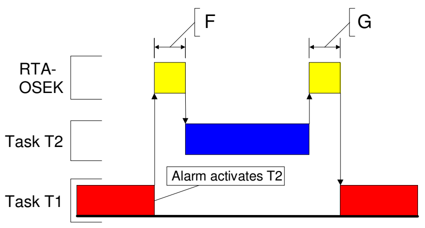

Figure 4: Task Activation from an Alarm

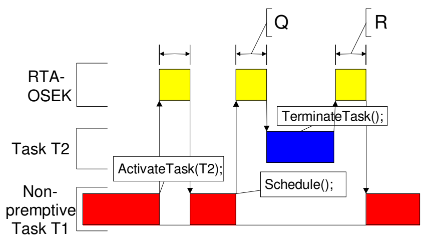

Figure 5: Non-Premptive Task Calls Schedule()

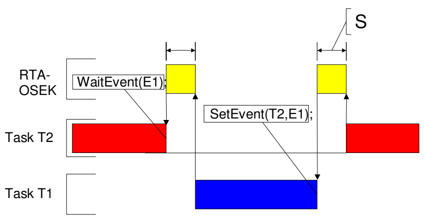

Figure 7: Waiting Task Activated by SetEvent()

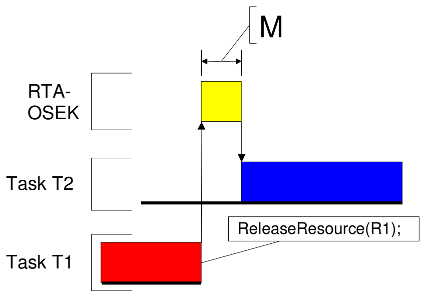

Figure 6: Blocked Task Activated by ReleaseResource()

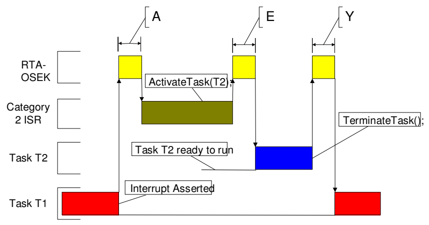

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	136	202	226	142	204	242
Figure 1: D	Heavy, Basic/Extended	267	325	350	335	331	354
ChainTask	Light, Basic	313	402	480	322	415	516
Figure 2: J	Heavy, Basic/Extended	777	939	1043	856	947	1062
Pre-emption	Light, Basic	253	330	437	261	338	470
Figure 1: C	Heavy, Basic/Extended	392	453	576	462	505	646
From idle task	Light, Basic	251	328	437	261	338	470
Figure 3: H	Heavy, Basic/Extended	390	451	572	462	505	646
Triggered by alarm	Light, Basic	410	480	597	418	492	630
Figure 4: F	Heavy, Basic/Extended	551	607	732	612	652	803
Schedule	Light, Basic	211	235	311	215	240	315
Figure 5: Q	Heavy, Basic/Extended	350	358	448	411	403	497
Release resource	Light, Basic	213	237	298	220	250	310
Figure 6: M	Heavy, Basic/Extended	352	360	435	416	413	492
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	797	779	941
From category 2 ISR	Light, Basic	236	337	392	316	338	401
Figure 8: E	Heavy, Basic/Extended	375	460	529	512	501	583

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	499	545	576	502	552	575
Figure 1: D	Heavy, Basic/Extended	628	657	690	668	667	693
ChainTask	Light, Basic	741	810	887	740	813	928
Figure 2: J	Heavy, Basic/Extended	1542	1673	1778	1596	1672	1790
Pre-emption	Light, Basic	521	594	685	524	596	717
Figure 1: C	Heavy, Basic/Extended	644	714	825	728	764	890
From idle task	Light, Basic	519	592	684	524	596	717
Figure 3: H	Heavy, Basic/Extended	642	712	822	728	764	890
Triggered by alarm	Light, Basic	673	754	840	677	754	870
Figure 4: F	Heavy, Basic/Extended	798	878	977	874	915	1040
Schedule	Light, Basic	470	490	562	479	495	579
Figure 5: Q	Heavy, Basic/Extended	597	612	696	683	672	753
Release resource	Light, Basic	482	503	560	484	507	569
Figure 6: M	Heavy, Basic/Extended	609	625	694	688	684	743
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1056	1040	1199
From category 2 ISR	Light, Basic	858	956	1015	949	972	1024
Figure 8: E	Heavy, Basic/Extended	984	1075	1146	1148	1146	1196

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	677	717	744	663	706	745
Figure 1: D	Heavy, Basic/Extended	786	809	830	812	809	832
ChainTask	Light, Basic	1069	1142	1296	1109	1157	1264
Figure 2: J	Heavy, Basic/Extended	2041	2128	2294	2103	2153	2276
Pre-emption	Light, Basic	732	807	966	776	821	935
Figure 1: C	Heavy, Basic/Extended	855	930	1105	964	1002	1117
From idle task	Light, Basic	732	807	966	773	818	932
Figure 3: H	Heavy, Basic/Extended	855	930	1105	961	999	1114
Triggered by alarm	Light, Basic	958	1024	1183	999	1039	1149
Figure 4: F	Heavy, Basic/Extended	1084	1150	1319	1186	1215	1330
Schedule	Light, Basic	531	549	624	533	539	621
Figure 5: Q	Heavy, Basic/Extended	654	672	760	721	716	801
Release resource	Light, Basic	700	707	781	749	757	816
Figure 6: M	Heavy, Basic/Extended	823	830	917	937	934	996
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1334	1333	1485
From category 2 ISR	Light, Basic	901	1003	1056	979	987	1062
Figure 8: E	Heavy, Basic/Extended	1018	1120	1192	1166	1163	1242

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		176	176	192	176	176	192
BCC1 lightweight, floating-point		192	192	208	192	192	208
BCC1 heavyweight, integer		288	288	304	288	288	304
BCC1 heavyweight, floating-point		288	288	304	288	288	304
BCC2 lightweight, integer		n/a	192	208	n/a	192	208
BCC2 lightweight, floating-point		n/a	192	208	n/a	192	208
BCC2 heavyweight, integer		n/a	304	320	n/a	304	320
BCC2 heavyweight, floating-point		n/a	304	320	n/a	304	320
ECC1 heavyweight, integer		n/a	n/a	n/a	336	336	304
ECC1 heavyweight, floating-point		n/a	n/a	n/a	336	336	304
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	304
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	304
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		192	192	192	192	192	192
BCC1 lightweight, floating-point		208	208	208	208	208	208
BCC1 heavyweight, integer		304	304	304	304	304	304
BCC1 heavyweight, floating-point		304	304	304	304	304	304
BCC2 lightweight, integer		n/a	208	208	n/a	208	208
BCC2 lightweight, floating-point		n/a	208	208	n/a	208	208
BCC2 heavyweight, integer		n/a	320	320	n/a	320	320
BCC2 heavyweight, floating-point		n/a	320	320	n/a	320	320
ECC1 heavyweight, integer		n/a	n/a	n/a	304	304	304
ECC1 heavyweight, floating-point		n/a	n/a	n/a	304	304	304
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	304
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	304

Timing

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
Pre- and Post-Task hooks not used						
Task type						
BCC1 lightweight, integer	224	224	224	224	224	224
BCC1 lightweight, floating-point	240	240	240	240	240	240
BCC1 heavyweight, integer	336	336	336	336	336	336
BCC1 heavyweight, floating-point	336	336	336	336	336	336
BCC2 lightweight, integer	n/a	240	240	n/a	240	240
BCC2 lightweight, floating-point	n/a	240	240	n/a	240	240
BCC2 heavyweight, integer	n/a	336	336	n/a	336	336
BCC2 heavyweight, floating-point	n/a	336	336	n/a	336	336
ECC1 heavyweight, integer	n/a	n/a	n/a	416	416	352
ECC1 heavyweight, floating-point	n/a	n/a	n/a	416	416	352
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	352
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	352
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	224	224	240	224	224	240
BCC1 lightweight, floating-point	240	240	256	240	240	256
BCC1 heavyweight, integer	336	336	352	336	336	352
BCC1 heavyweight, floating-point	336	336	352	336	336	352
BCC2 lightweight, integer	n/a	240	256	n/a	240	256
BCC2 lightweight, floating-point	n/a	240	256	n/a	240	256
BCC2 heavyweight, integer	n/a	336	352	n/a	336	352
BCC2 heavyweight, floating-point	n/a	336	352	n/a	336	352
ECC1 heavyweight, integer	n/a	n/a	n/a	352	352	368
ECC1 heavyweight, floating-point	n/a	n/a	n/a	352	352	368
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	368
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	368

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	224	224	224	224	224	224
BCC1 lightweight, floating-point	240	240	240	240	240	240
BCC1 heavyweight, integer	336	336	336	336	336	336
BCC1 heavyweight, floating-point	336	336	336	336	336	336
BCC2 lightweight, integer	n/a	240	240	n/a	240	240
BCC2 lightweight, floating-point	n/a	240	240	n/a	240	240
BCC2 heavyweight, integer	n/a	336	336	n/a	336	336
BCC2 heavyweight, floating-point	n/a	336	336	n/a	336	336
ECC1 heavyweight, integer	n/a	n/a	n/a	416	416	352
ECC1 heavyweight, floating-point	n/a	n/a	n/a	416	416	352
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	352
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	352
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	224	224	240	224	224	240
BCC1 lightweight, floating-point	240	240	256	240	240	256
BCC1 heavyweight, integer	336	336	352	336	336	352
BCC1 heavyweight, floating-point	336	336	352	336	336	352
BCC2 lightweight, integer	n/a	240	256	n/a	240	256
BCC2 lightweight, floating-point	n/a	240	256	n/a	240	256
BCC2 heavyweight, integer	n/a	336	352	n/a	336	352
BCC2 heavyweight, floating-point	n/a	336	352	n/a	336	352
ECC1 heavyweight, integer	n/a	n/a	n/a	352	352	368
ECC1 heavyweight, floating-point	n/a	n/a	n/a	352	352	368
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	368
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	368

5 Inline Interrupt Control API Calls

The RTA-OSEK Component for the MPC55XX/Diab supports two variations of the OSEK interrupt handling API calls. In addition to the API calls contained within the RTA-OSEK run-time libraries, inline versions are also supported. Using these inline versions will result in faster code and a reduced Context Save Area usage. The inline versions of these API calls are all have the “os” prefix.

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

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

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

6 Version 5.0.5

6.1 Variants

The supported target cores in the RTA-OSEK GUI are the e200z1 (MPC5514, MPC5516), e200z3 (MPC5533, MPC5534, SPC563M), e200z4 (MPC5643L, MPC5644A, SPC564A), e200z6 (MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, MPC5567), e200z7 (MPC5674F) and the MPC55xx Generic.

6.2 Kernel Linking

If an attempt to link a v5.0.5 OIL file with a v5.04 kernel library is made, a linker error will be seen. This is due to incompatible difference between v5.04 and v5.05 kernels.

7 Version 5.0.4

7.1 Variants

The supported target cores in the RTA-OSEK GUI are the e200z1 (MPC5514, MPC5516), e200z3 (MPC5533, MPC5534, SPC563M), e200z4 (MPC5643L, MPC5644A, SPC564A), e200z6 (MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, MPC5567), e200z7 (MPC5674F) and the MPC55xx Generic.

7.2 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.

8 Version 5.0.3

8.1 Variants

The supported targets in the RTA-OSEK GUI are the MPC5533, MPC5534, MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, MPC5567, MPC5514 (Z1 core only), MPC5516 (Z1 core only), SPC563M, MPC5674F and MPC55xx Generic.

8.2 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.

8.3 Issues with Compiler Version 5.7.0.0 and Diab5700-1_wind00175250 Patch

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

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

9 Version 5.0.2

9.1 Variants

The supported targets in the RTA-OSEK GUI have been extended from the MPC5533, MPC5534, MPC5553, MPC5554, MPC5561, MPC5565, MPC5566, MPC5567, MPC5514 (Z1 core only), MPC5516 (Z1 core only) and generic MPC55xx to additionally include the SPC563M and MPC5674.

9.2 Kernel Linking

If an attempt to link a v5.0.2 OIL file with a v5.01 kernel library is made, a linker error will be seen. This is due to incompatible difference between v5.01 and v5.02 kernels.

10 Version 5.0.1

10.1 Variants

The supported targets in the RTA-OSEK GUI have been extended from the MPC5534, MPC5553, MPC5554, MPC5565, MPC5566, MPC5567 and generic MPC55xx to additionally include the MPC5533, MPC5514 (Z1 core only), MPC5516 (Z1 core only) and MPC5561.

10.2 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 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.

10.3 Linker Files

Linker command files (.dld) must be modified from the previous release to add the new memory sections (os_pir2 and os_pnir2). Refer to the Binding Manual Diab55xx for details.

10.4 Kernel Linking

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

11 Version 5.0.0

11.1 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 3.1.7.

12 Compatibility with Pre-v5 Kernels

12.1 Updating the application version

To convert an existing v3.x OIL configuration file to v5.0.1, load the file into the v5 RTA-OSEK GUI, select the ‘OS Configuration’ option in the ‘Application’ menu and change the ‘Kernel Version’ to v5.01. When the OIL configuration file is saved it will then use the v5.0.1 format and the v5.01 kernel libraries. This process can be reversed to move back to earlier kernel versions. However, if an attempt to link a v5.0.1 OIL file with a v5.00 kernel library is made, a linker error will be seen.

12.2 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.