RTA-OSEK

Binding Manual: TMS470/ADS

Contact Details

ETAS Group

www.etasgroup.com

Germany

ETAS GmbH
Borsigstraße 14
70469 Stuttgart

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

www.etas.de

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

Japan

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

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

www.etas.co.jp

France

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

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

www.etas.fr

Korea

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

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

www.etas.co.kr

Great Britain

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

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

www.etas-uk.net

Copyright Notice

© 2001 - 2005 LiveDevices Ltd. All rights reserved.

Version: RM00067-002

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

Disclaimer

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

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

Trademarks

RTA-OSEK and LiveDevices are trademarks of LiveDevices Ltd.

Windows and MS-DOS are trademarks of Microsoft Corp.

OSEK/VDX is a trademark of Siemens AG.

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

Contents

1 About this Guide 7

1.1 Who Should Read this Guide? 7

1.2 Conventions..7

2 Toolchain Issues 9

2.1 Compiler..9

2.1.1 Using the THUMB/ARM Assembler Instruction Set 10

2.2 Assembler 10

2.2.1 Software and Hardware Vector Tables 10

2.3 Linker/Locator 11

2.3.1 Sections 12

2.3.2 Locating the IRQIVEC and FIQIVEC Registers....................12

2.4 Debugger..13

3 Target Hardware Issues...15

3.1 Interrupts 15

3.1.1 Interrupt Levels 15
3.1.2 Interrupt Vectors 15
3.1.3 Category 1 Handlers..16
3.1.4 Category 2 Handlers..16
3.1.5 Vector Table Issues 16
3.1.6 Interrupt Priority and Category Constraints......................17
3.1.7 Software Vectoring 17
3.1.8 Hardware Vectoring 20
3.1.9 Initializing FIRQPR Register 21
3.1.10 Reset Vector 22
3.1.11 VIM Channel Arbitration Order 22
3.2 Control of FIQ Interrupts..22
3.2.1 RTA-OSEK and User FIQ Bit Manipulation........................22
3.3 CPU Operating Modes..23
3.4 Register Settings..24
3.5 Stack Usage 24
3.5.1 Number of Stacks..24
3.5.2 Stack Usage within API Calls..25
3.6 IRQ Stack Usage 25
4 Parameters of Implementation 27
4.1 Functionality..27
4.2 Hardware Resources..28
4.2.1 ROM and RAM Overheads..28
4.2.2 ROM and RAM for OSEK OS Objects...............................29
4.2.3 Size of Linkable Modules 34
4.2.4 Reserved Hardware Resources..47
4.3 Performance..47
4.3.1 Execution Times for RTA-OSEK API Calls..........................47
4.3.2 OS Start-up Time..57
4.3.3 Interrupt Latencies..57
4.3.4 Task Switching Times 58
4.4 Configuration of Run-time Context..61
5 Extended Interrupt Control Functions 65

5.1 Globally Disabling and Enabling IRQ Interrupts 65

5.2 Globally Disabling and Enabling FIQ Interrupts...........................65

5.3 Disabling and Enabling VIM Channels 66

5.4 Rules for Using the Extended Interrupt Control Functions 67

5.4.1 Tasks and ISRs Must Preserve F and I bit Settings 67

5.4.2 Do Not Use RTA-OSEK APIs when the F and I bits Have Been Changed...68

5.4.3 The `OSSuspendFIQ()` and `OSResumeFIQ()` are Not Re-Entrant 69

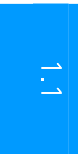
5.4.4 The `OSSuspendIRQ()` and `OSResumeIRQ()` are Not Re-Entrant 69

5.4.5 The `OSSuspendVIMChannel()` and `OSResumeVIMChannl()` are Not Re-Entrant 69

6 Implementing SWI Handlers..70

6.1 Handling an SWI 70

6.2 Generating an SWI 71



1 About this Guide

This guide provides port specific information for the TMS470/ADS implementation of LiveDevices' RTA-OSEK.

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

1.1 Who Should Read this Guide?

It is assumed that you are a developer. You should read this guide if you want to know low-level technical information to integrate the RTA-OSEK Component into your application.

1.2 Conventions

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

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

In this guide you'll see that program code, header file names, C type names, C functions and RTA-OSEK API call names all appear in the `courier` typeface. When the name of an object is made available to the programmer the name also appears in the `courier` typeface, so, for example, a task named Task1 appears as a task handle called `Task1`.

2 Toolchain Issues

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

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

2.1 Compiler

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

Vendor	ARM Ltd.
Compiler	RealView Developer Suite
Version	RVCT2.2 (Build 435)

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

Option	Description
`--bi`	Compile for big-endian byte access format

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
`--bi`	Compile for big-endian byte access format
`-O2`	Maximum optimization level
`--cpu 4T`	Processor version ARM v4 with long multiply and THUMB
`--apcs /inter/noswst`	Provide interworking support/No software stack checking
`--thumb`	Generate THUMB code

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

Option	Description
`-g`	Debugging support

2.1.1 Using the THUMB/ARM Assembler Instruction Set

The libraries support applications using either the THUMB or the ARM instruction set. If C files are compiled with the ARM instruction set then inter-working support should be used (i.e. the `--apcs /inter` compiler and assembler options). To reduce the amount of code memory used, the RTA-OSEK run-time libraries have mainly been compiled using the THUMB instruction set.

2.2 Assembler

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

Vendor	ARM Ltd.
Assembler	RealView Developer Suite
Version	RVCT2.2 (Build 435)

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

Option	Description
`--bi`	Assemble for big-endian byte access format

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
`--bi`	Assemble for big-endian byte access format
`--cpu 4T`	Processor version ARM v4 with long multiply and THUMB
`--apcs /noswst`	No software stack checking

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

Option	Description
`-g`	Debugging support

2.2.1 Software and Hardware Vector Tables

To support both hardware and software vectoring, in addition to `osgen.s`, RTA-OSEK also generates files containing vector tables. File `osswvect.s` contains the vector table and ISR information table for software vectoring. File `oshwvect.s` contains the vector table and ISR information table for

hardware vectoring. One of osswvect.s or oshwvect.s should be assembled and linked with your application. The same assembly options used for osgen.s should be used for osswvect.s or oshwvect.s.

2.3 Linker/Locator

The compulsory linker/locator options for an RTA-OSEK application are shown in the following table:

Option	Description
--noremove	Do not remove unused sections. Without this option the stack and heap sections used in the application will be removed.

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
os_pird	ROM	RTA-OSEK initialization data
os_vectbl	ROM	Vector table if generated by RTA-OSEK GUI
os_pir	RAM	RTA-OSEK initialized data
os_pur	RAM	RTA-OSEK uninitialized data
os_text	ROM	RTA-OSEK code
os_constdata	ROM	RTA-OSEK read-only data
os_data	RAM	RTA-OSEK initialized data
os_bss	RAM	RTA-OSEK zeroed data
os_ivec	RAM	Location of the IRQIVEC and FIQIVEC registers

In some cases, sections produced by the linker must be located according to special constraints. The following table indicates which sections must be located with which particular constraints:

Sections	Constraints
os_vectbl	Must be located at address 0x00000004

The following compiler run-time library functions are required by the RTA-OSEK Component:

C Library Functions	Description
__ARM_call_via_r0	Indirect call helper
__ARM_call_via_r1	Indirect call helper
__ARM_call_via_r4	Indirect call helper
__ARM_call_via_r5	Indirect call helper

2.3.1 Sections

To make it easier for you to distinguish RTA-OSEK code and data from application code and data, RTA-OSEK does not use the default compiler generated sections. All RTA-OSEK sections are prefixed os_.

When the RTA-OSEK Component libraries were compiled the following pragmas were in effect:

```
#pragma arm section code = "os_text"
#pragma arm section rwdata = "os_data"
#pragma arm section rodata = "os_constdata"
#pragma arm section zidata = "os_bss"
```

As a result RTA-OSEK uses sections as shown in the following table:

Default Section	RTA-OSEK Section	Section Contents
||.text||	os_text	Code
||.constdata||	os_constdata	Constant data
||.data||	os_data	Initialized data
||.bss||	os_bss	Zeroed data

The sections os_text, os_constdata, os_data and os_bss should be treated in the same way as their equivalent defaults in terms of linking.

RTA-OSEK also uses some additional RAM sections called os_pur and os_pir. These sections are initialized by the RTA-OSEK Component within the StartOS() API call (unlike the standard C RAM variable sections os_data and os_bss, which are initialized in the application start-up code). These sections may be marked UNINIT in the linker command file, which prevents the contents of the section being additionally initialized in the application start-up code.

2.3.2 Locating the IRQIVEC and FIQIVEC Registers

RTA-OSEK uses some definitions in a library module called osxivec.o to allow the software de-multiplexers to locate the IRQIVEC and FIQIVEC registers. The library module osxivec.o must be located at address 0xFFFFFE00. The memory region occupied by osxivec.o must not be initialized (i.e. use the UNINIT attribute). The following code in a linker command file will achieve this (see also the linker command file in the example application):

```
A_LOAD_REGION <start-address>
{
  ; One or more execution regions.

  ; Locate osxivec.o at the address of IRQIVEC.
  IVEC_LOC 0xFFFFFE00 UNINIT
  {
        osxivec.o (+BSS)
  }
}
```

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 (`.axf`) file. Please refer to the debugger documentation for further details on its support for ORTI.

3 Target Hardware Issues

3.1 Interrupts

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

3.1.1 Interrupt Levels

In RTA-OSEK interrupts are allocated an Interrupt Priority Level (IPL). This is a processor independent abstraction of the interrupt priorities that are available on the target hardware. You can find out more about IPLs in the *RTA-OSEK User Guide*. The hardware interrupt controller is explained in the *Design Specification for TMS470PVF241PN, TMS470PVF242PN and TMS470PVF241PZ*.

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

IPL Value	CPSR [7:6]	Description
0	00	User level
0	01	User level - see section 3.2.
1	10	Category 1 and 2 interrupts
2	11	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	Description	Legality
0x04	Undefined instructions	Category 1
0x08	SWI	Category 1
0x0C	Prefetch Abort	Category 1
0x10	Data Abort	Category 1
0x18	IRQ (General Interrupt)	Single Category 1 or 2 (S/W vectoring only)
0x1C	FIQ (Fast Interrupt)	Single Category 1
0x20	VIM channel 0	Category 1 or 2
0x24	VIM channel 1	Category 1 or 2
0x28	VIM channel 2	Category 1 or 2
…	…	…

Vector	Description	Legality
0x118	VIM channel 62	Category 1 or 2
0x11C	VIM channel 63	Category 1 or 2

The valid base addresses for the vector table are:

Base Address	Notes
0x00000000	The CPU vector table must be located at address 0x00000000.

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 ARM Ltd. C compiler can generate appropriate interrupt handling code for a C function decorated with the `[__irq] void` 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 (VVV represents the 3 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label	Notes
0xVVV	os_wrapper_VVV	Only used for H/W vectoring
eg : 0x2C	os_wrapper_02C	

3.1.6 Interrupt Priority and Category Constraints

Important: The following rules about interrupt priority and Category apply.

IRQ interrupts are priority 1. All other interrupt sources are priority 2.

IRQ interrupts may be Category 1 or 2. All other interrupts must be Category 1.

If a VIM channel is bound to a priority 1 ISR then RTA-OSEK assumes that the VIM channel has been configured to generate an IRQ interrupt.

If a VIM channel is bound to a priority 2 ISR then RTA-OSEK assumes that the VIM channel has been configured to generate an FIQ interrupt.

3.1.7 Software Vectoring

Software vectoring makes use of the VIM's `IRQIVEC` and `FIQIVEC` registers to identify which VIM channel is the source of an interrupt.

The osswvect.s Assembly File

The `osswvect.s` assembly file generated by RTA-OSEK contains the following tables used to support software vectoring:

- `os_vec_start` (in section `os_vectbl`) – the CPU's vector table.
- `os_sw_vectors` (in section `os_pid`) – an array of pointers to interrupt handlers. `os_sw_vectors[n]` points to the interrupt handler for VIM channel `n-1`.
- `os_isr_info` (in section `os_pid`) – an array of pointers to Category 2 ISR information. `os_isr_info[n]` points to the ISR information for VIM channel `n-1`.

The CPU Vector Table

For a Category 1 ISR bound to any interrupt other than IRQ and FIQ, the CPU vector table entry for the interrupt branches directly to the Category 1 ISR function.

Considering IRQ and FIQ interrupts, the following scenarios are possible:

- Single Category 1 ISR bound to the FIQ (0x1C) vector. In this case the entry for offset 0x1C in the CPU vector table will branch directly to the Category 1 ISR function.
- Priority 2 (i.e. FIQ) Category 1 ISRs bound to VIM vectors (0x20-0x11C). In this case the entry for offset 0x1C in the CPU vector table will branch to the RTA-OSEK software de-multiplexer `os_fiq_entrym`.
- Single Category 2 ISR bound the IRQ (0x18) vector. In this case the entry for offset 0x18 in the CPU vector table will branch to the RTA-OSEK software de-multiplexer `os_irq_entrys`.
- Priority 1 (i.e. IRQ) Category 1 or 2 ISRs bound to VIM vectors (0x20-0x11C). In this case the entry for offset 0x18 in the CPU vector table will branch to the RTA-OSEK software de-multiplexer `os_irq_entrym`.

Important: when software vectoring is used and a single FIQ interrupt is in use, better performance is obtained if the ISR for the FIQ interrupt is bound directly to the FIQ vector. Likewise if a single IRQ interrupt is in use, better performance is obtained if the ISR for the IRQ interrupt is bound directly to the IRQ vector.

Software De-multiplexers

The 3 software de-multiplexers behave as follows:

`os_fiq_entrym`

Save the link register, `SPSR_fiq` and registers that may be corrupted by C functions on the stack.

Read `FIQIVEC` to find out which VIM channel caused the interrupt.

Call the Category 1 interrupt handler pointed to by `os_sw_vectors[FIQIVEC]`.

Restore registers from the stack.

Return from interrupt.

`os_irq_entrys`

Save the link register, SPSR_irq and registers that may be corrupted by C functions on the stack.

Switch to SVC mode.

Set register `r0` to 0.

Call RTA-OSEK's internal Category 2 interrupt handler, `os_wrapper`.

Switch back to IRQ mode.

Restore registers from the stack.

Return from interrupt.

os_irq_entrym

Save the link register, SPSR_irq and registers that may be corrupted by C functions on the stack.

Switch to SVC mode.

Read IRQIVEC to find out which VIM channel caused the interrupt.

Store the value read from IRQIVEC in register r0.

Call the interrupt handler pointed to by os_sw_vectors[IRQIVEC]. This is the ISR function for Category 1 ISRs and RTA-OSEK's internal Category 2 interrupt handler, os_wrapper, for Category 2 ISRs.

Switch back to IRQ mode.

Restore registers from the stack.

Return from interrupt.

Supplying Your Own Software De-multiplexers

You may supply your own software de-multiplexers to override the ones supplied with RTA-OSEK. If you do, the following rules must be obeyed:

- The software de-multiplexers are responsible for returning from interrupts.
- If an IRQ interrupt is bound to a Category 2 ISR then RTA-OSEK's internal Category 2 interrupt handler, os_wrapper, must be called. When os_wrapper is called the following must hold:
    - The processor must be in SVC mode.
    - os_wrapper may re-enable IRQ interrupts – so the de-multiplexers that call it must be re-entrant (i.e. at least the link and SPSR registers must be stored on the stack).
    - os_wrapper is a C function so any registers that may be corrupted by a C function must have been preserved.
    - When os_wrapper is called and there is a single ISR bound to vector 0x18, register r0 must contain the value 0.
    - When os_wrapper is called for an interrupt bound to a VIM vector, then register r0 must contain the number of the channel that caused the interrupt plus 1. This value can be read from IRQIVEC.

If you do supply your own software de-multiplexers the interrupt latencies quoted later in this manual may no longer be correct.

Category 1 ISR Functions

If a Category 1 ISR function is entered via a software de-multiplexer then the ISR function should be an ordinary C function. If the CPU vector table

branches directly to the ISR function then the ISR function must take care of returning from interrupts – e.g. by using the `__irq` C function modifier.

3.1.8 Hardware Vectoring

Hardware vectoring makes use of the VIM's `IRQVECREG` and `FIQVECREG` registers to determine the correct interrupt handler when a VIM channel causes an interrupt.

The oshwvect.s Assembly File

The `oshwvect.s` assembly file generated by RTA-OSEK contains the following tables used to support hardware vectoring:

- `os_vec_start` (in section `os_vectbl`) – the CPU's vector table.
- `os_vim_vectors` (in section `os_pid`) – an array of pointers to interrupt handlers. `os_vim_vectors[n]` points to the interrupt handler for VIM channel `n-1`. This array must be used to initialize the VIM Vector Offset Registers.
- `os_isr_info` (in section `os_pid`) – an array of pointers to Category 2 ISR information. `os_isr_info[n]` points to the ISR information for VIM channel `n-1`.

The CPU Vector Table

For a Category 1 ISR bound to any interrupt other than IRQ and FIQ, the CPU vector table entry for the interrupt branches directly to the Category 1 ISR function.

Considering IRQ and FIQ interrupts, the following scenarios are possible:

- Single Category 1 ISR bound to the FIQ (0x1C) vector. In this case the entry for offset 0x1C in the CPU vector table will branch directly to the Category 1 ISR function.
- Priority 2 (i.e. FIQ) Category 1 ISRs bound to VIM vectors (0x20-0x11C). In this case the entry for offset 0x1C in the CPU vector table will contain the instruction `LDR pc,[pc,#-0x1B0]`. This will cause the CPU to jump to the interrupt handler specified in `FIQVECREG` – i.e. the VIM channel specific handler from the VIM vector table.
- Single Category 2 ISR bound the IRQ (0x18) vector. **Not allowed for hardware vectoring.**
- Priority 1 (i.e. IRQ) Category 1 or 2 ISRs bound to VIM vectors (0x20-0x11C). In this case the entry for offset 0x18 in the CPU vector table will contain the instruction `LDR pc,[pc,#-0x1B0]`. This will cause the CPU to jump to the interrupt handler specified in `IRQVECREG` – i.e. the VIM channel specific handler from the VIM vector table.

Important: when hardware vectoring is used an ISR must not be bound directly to the IRQ vector (vector 0x18). With hardware vectoring there is no advantage to binding an ISR directly to the IRQ vector since all of the de-multiplexing of VIM channels is done in hardware. Since RTA-OSEK generates vector tables for software and hardware interrupts at the same time, and binding an ISR to the IRQ vector is valid for software vectoring, an error is not generated when an ISR is bound directly to the IRQ vector.

The VIM Vector Table

RTA-OSEK creates a VIM vector table called `os_vim_vectors`. This table must be copied into the VIM Offset Vector Registers at address 0xFFF82000 before `StartsOS()` is called. `os_vim_vectors` contains 65 32bit entries: one entry for the phantom vector and then one entry for each VIM channel. See the example application for an example of how to initialize the VIM Offset Vector Registers.

The VIM vector table entry for a Category 1 interrupt is the address of the Category 1 ISR. The VIM vector table entry for a Category 2 interrupt is the address of the outer wrapper for the VIM vector. For example, if a Category 2 ISR is bound to VIM channel 0, which is VIM vector 0x20, then the VIM vector table entry for VIM channel 0 will be the address of the routine `os_wrapper_020`. Routine `os_wrapper_020` is a wrapper specific to VIM vector 0x20 and will be found in the file `oshwvect.c`.

Important: although the VIM interrupt controller can support 64 channels, the implementation in the TMS470PVF24xPx only supports 32 channels. Therefore only the first 33 entries of `os_vim_vectors` (phantom vector plus 32 channels) should be used to initialize the VIM Offset Vector Registers when using a TMS470PVF24xPx.

Category 1 ISR Functions

With hardware vectoring all Category 1 ISRs are branched to directly from the CPU vector table. Thus all Category 1 ISR functions must take care of returning from interrupts – e.g. by using the `__irq` C function modifier.

3.1.9 Initializing FIRQPR Register

The `FIRQPR` register must be initialized so that all VIM channels bound to priority 2 ISRs (i.e. FIQ interrupts) are configured to generate FIQ interrupts. The constants `OS_FIRQPR_LO_INIT` and `OS_FIRQPR_HI_INIT` are automatically created by RTA-OSEK in the header file `osekcomn.h` for this purpose. `OS_FIRQPR_LO_INIT` should be written to `FIRQPR [31:0]` and `OS_FIRQPR_HI_INIT` should be written to `FIRQPR [63:32]`.

3.1.10 Reset Vector

The CPU vector table generated by RTA-OSEK does not include an entry for the reset vector. The application must setup the reset vector to branch to code that performs the start-up operation required for the ARM C compiler and sets up the stack pointers for the CPU modes used. An example is provided in the example application.

3.1.11 VIM Channel Arbitration Order

Lower numbered VIM channels have a higher arbitration order than higher numbered VIM channels. This means that if IRQ interrupts are pending on VIM channel numbers m and n, where m < n, then when IRQ interrupts are enabled, the interrupt on channel number m will be serviced before the interrupt on channel n – irrespective of whether Category 1 or 2 ISRs are bound to VIM channel numbers m and n.

Likewise if FIQ interrupts are pending on VIM channel numbers m and n, where m < n, then when FIQ interrupts are enabled, the interrupt on channel number m will be serviced before the interrupt on channel n.

3.2 Control of FIQ Interrupts

To allow the application maximum control over FIQ interrupts, RTA-OSEK is very careful to avoid manipulating bit 6 of the CPSR register that is used to disable FIQ interrupts. Except for when the extended interrupt control functions are used – see section 5 - the only time that RTA-OSEK will change the value of bit 6 of the CPSR register is when the OSEK API functions:

```
DisableAllInterrupts()
EnableAllInterrupts()
SuspendAllInterrupts()
ResumeAllInterrupts()
```

are called. `DisableAllInterrupts()` and `SuspendAllInterrupts()` will set bit 6 of the CPSR to disable FIQ interrupts. `EnableAllInterrupts()` and `ResumeAllInterrupts()` will clear bit 6 of the CPSR if it was clear before the corresponding call to `DisableAllInterrupts()` or `SuspendAllInterrupts()`.

3.2.1 RTA-OSEK and User FIQ Bit Manipulation

RTA-OSE runs at one of three interrupt priority levels. These are:

Level	CPSR [7:6]	Description
0	00	User level – in task code
0	01	User level – in task code

1	10	Operating system level – in IRQ interrupt code
2	11	Above operating system level – in FIQ interrupt code

Most RTA-OSEK API functions must not be called from ISRs that are running above operating system interrupt level. A side effect of RTA-OSEK not manipulating CPSR bit 6 is that if the application disables FIQ interrupts by setting bit 6 of CPSR and then an API call is made from a priority 1 (i.e. IRQ) Category 2 ISR, RTA-OSEK may believe that it is being called from above operating system level. This will cause the OSEK error hook to be entered if the application is in extended build.

Important: The following steps can be taken to avoid this problem.

Always enable FIQ interrupts before `StartOS()` is called, or better in the OSEK start-up hook.

Never use the extended interrupt control functions to disable FIQ interrupts without also disabling IRQ interrupts. Or, never call RTA-OSEK API functions from inside Category 2 ISRs in extended build when FIQ interrupts have been disabled.

3.3 CPU Operating Modes

A TMS470 CPU can use up to six different modes in an application, each with their own stack. The RTA-OSEK Component always expects to run in SVC mode (i.e. all tasks and Category 2 ISRs execute in SVC mode).

When a Category 2 interrupt occurs, the RTA-OSEK Component switches from IRQ mode to SVC mode before processing the interrupt. This minimizes the worst-case stack requirements.

Category 1 interrupts, which operate outside of the RTA-OSEK Component, can be configured to use the following CPU operating modes; SVC by using the SWI, FIQ, IRQ, Abort and Undefined.

Important: All stack pointers must be initialized before use.

Important: RTA-OSEK requires that the SVC stack section is called `SVC_STACK`.

The example application demonstrates a suggested method for stack pointer initialization. The start-up code, `init.s`, declares the each stack section with the lengths defined by `STACK_LEN_<mode>` values. The linker command file, `link_flash.scf`, locates the stack's sections in memory. The linker implicitly defines labels at the top of each stack section, e.g. `Image$$SVC_STACK$$ZI$$Limit` for the `SVC_STACK` section.

3.4 Register Settings

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

Register	Setting
`FIRQPR`	Set so that all VIM channels bound to priority 2 ISRs are configured to be FIQ. The constants `OS_FIRQPR_LO_INIT` and `OS_FIRQPR_HI_INIT` can be used see section 3.1.9.
`REQMASK`	Set so that all VIM channels bound to ISRs are enabled.
`CPSR [7]`	Set to 0 to disable IRQ interrupts - see section 3.2.
`VIM Offset Vector Registers`	Only for H/W vectoring. Must be initialized from the table `os_vim_vectors` - see section 3.1.8.

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

Register	Use
`CPSR [7:0]`	Used to control IRQ and FIQ interrupts and processor mode - see section 3.2.
`IRQIVEC`	Only for S/W vectoring. Read to determine which VIM channel caused an interrupt.
`FIQIVEC`	Only for S/W vectoring. Read to determine which VIM channel caused an interrupt.
`IRQVECREG`	Only for H/W vectoring. Read to determine the interrupt handler for a VIM channel.
`FIQVECREG`	Only for H/W vectoring. Read to determine the interrupt handler for a VIM channel.
`REQMASK`	Only if the extended interrupt control functions are used - see section 5.

3.5 Stack Usage

3.5.1 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: `typedef unsigned long`.

3.5.2 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): 40

Timing

API max usage (bytes): 40

Extended

API max usage (bytes): 56

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

3.6 IRQ Stack Usage

The stack usage reported by RTA-OSEK is SVC stack usage. In addition to the SVC stack usage reported by RTA-OSEK, each Category 2 IRQ interrupt also uses 36 bytes of IRQ stack.

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	**TMS470PVF241PN**
Clock speed (MHz)	15
Code memory	Internal Flash
Read-only data memory	Internal RAM
Read-write data memory	Internal RAM

4.1 Functionality

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

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

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

4.2 Hardware Resources

4.2.1 ROM and RAM Overheads

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	26	26	26	26	26	26
	ROM	156	156	156	156	156	156
COM overhead	RAM	8	8	8	8	8	8
	ROM	14	14	14	14	14	14

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	46	46	46	46	46	46
	ROM	228	228	228	228	228	228
COM overhead	RAM	8	8	8	8	8	8
	ROM	14	14	14	14	14	14

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
OS overhead	RAM	64	64	64	64	64	64
	ROM	274	274	274	274	274	274
COM overhead	RAM	8	8	8	8	8	8
	ROM	14	14	14	14	14	14

4.2.2 ROM and RAM for OSEK OS Objects

In addition to the base OS overhead, detailed in Section 4.2.1, each OSEK OS object requires ROM and/or RAM. RTA-OSEK provides additional sub-task types for each task type in OSEK (basic and extended), determined by the offline configuration tools. They are as follows:

OSEK Class	Termination	Arithmetic
BCC1	Lightweight	Integer or Floating-Point
BCC1	Heavyweight	Integer or Floating-Point
BCC2	Light or Heavy	Integer or Floating-Point
ECC1	Heavyweight	Integer
ECC1	Heavyweight	Floating-Point
ECC2	Heavyweight	Integer
ECC2	Heavyweight	Floating-Point

The following tables give the ROM and/or RAM requirements (in bytes) for each OS object in the RTA-OSEK Component. (Note that the OSEK COM class was set to CCCA for systems without events, CCCB for systems with events. A default message of size 10 bytes was used for both CCCA and CCCB. The CCCB message size includes queued messages.)

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	36	36	36	36	36	36
BCC1 Heavyweight task	RAM	4	4	4	4	4	4
	ROM	40	40	40	40	40	40
BCC2 task	RAM	n/a	8	10	n/a	8	10
	ROM	n/a	44	52	n/a	44	52
ECC1, Integer task	RAM	n/a	n/a	n/a	140	140	140
	ROM	n/a	n/a	n/a	60	60	60
ECC1, floating-point task	RAM	n/a	n/a	n/a	142	142	142
	ROM	n/a	n/a	n/a	60	60	60
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	142
	ROM	n/a	n/a	n/a	n/a	n/a	68
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	144
	ROM	n/a	n/a	n/a	n/a	n/a	68
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	44	44	44	44	44	44
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	64	64	64	64	64	64
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	32	32	32	32	32	32
Counter	RAM	4	4	4	4	4	4
	ROM	44	44	44	44	44	44
Message	RAM	11	11	11	51	51	51
	ROM	20	20	20	56	56	56
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	52	52	52	52	52	52
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	56	64	n/a	56	64
ECC1, Integer task	RAM	n/a	n/a	n/a	152	152	152
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating-point task	RAM	n/a	n/a	n/a	154	154	154
	ROM	n/a	n/a	n/a	72	72	72
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	154
	ROM	n/a	n/a	n/a	n/a	n/a	80
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	156
	ROM	n/a	n/a	n/a	n/a	n/a	80

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	84	84	84	84	84	84
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	92	92	92	92	92	92
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	32	32	32	32	32	32
Counter	RAM	4	4	4	4	4	4
	ROM	44	44	44	44	44	44
Message	RAM	11	11	11	51	51	51
	ROM	20	20	20	56	56	56
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	16	16	16	16	16	16
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	64	72	n/a	64	72
ECC1, Integer task	RAM	n/a	n/a	n/a	156	156	156
	ROM	n/a	n/a	n/a	80	80	80
ECC1, floating-point task	RAM	n/a	n/a	n/a	158	158	158
	ROM	n/a	n/a	n/a	80	80	80
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	158
	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	160
	ROM	n/a	n/a	n/a	n/a	n/a	88
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	96	96	96	96	96	96
Category 2 ISR, floating-point	RAM	18	18	18	18	18	18
	ROM	104	104	104	104	104	104
Resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	36	36	36	36	36	36
Counter	RAM	4	4	4	4	4	4
	ROM	48	48	48	48	48	48
Message	RAM	11	11	11	51	51	51
	ROM	24	24	24	60	60	60
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Arrivalpoint (writable)	RAM	20	20	20	20	20	20
	ROM	20	20	20	20	20	20
Schedule	RAM	20	20	20	20	20	20
	ROM	44	44	44	44	44	44
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

4.2.3 Size of Linkable Modules

The RTA-OSEK Component is demand linked. This means that each API call is placed into a separately linkable module. The following sections list the module sizes (in bytes) for each API call in the 3 RTA-OSEK build types (standard, timing, and extended).

In some cases there are multiple variants of particular API calls. This is because the offline configuration of RTA-OSEK can determine when optimized versions of the API calls can be used. The smallest and fastest call will be selected. In these cases, modules sizes are given for each variant under the particular configuration of the RTA-OSEK Component for which the call is valid.

The call variants are as follows:

Variant	Description
1i	Idle task is only ECC task.
CCCA	OSEK COM class.
CCCB	OSEK COM class.
CLEx	Resource tests in Extended OS Status.
fp	ECC task uses floating-point.
H	Used for heavyweight termination only.

Variant	Description
Hook	Pre- and Post- Task hooks are used.
KL	API is called from OS level.
KL1i	API is called from OS level, idle task is only ECC task.
KL2	Activated taskset has one BCC2 task.
LExt	Used for lightweight termination in Extended Status.
ServiceID	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
Parameters	`ErrorHook` uses `GetServiceID` and `GetServiceParameters`.
NoHook	Pre- and/or Post- Task hooks are not used.
NS	No context switch is possible.
NS1i	No context switch is possible, idle task is only ECC task.
NS2	Activated taskset has one BCC2 task.
NSH	Chain from heavyweight task, not to higher priority.
NSL	Chain from lightweight task, not to higher priority.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.
Task	Resource is used only by tasks.

Standard

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Service name	Variant	Notes						
ActivateTask	SW	1	100	132	164	112	144	196
	NS		80	112	144	92	124	176
	KL	2	60	88	120	72	100	156

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	36	36	36	36	36	36
ChainTask	SWL	1, 8	80	132	160	100	140	188
	SWH	1, 9	112	156	184	124	164	212
	NSL	8	80	132	160	100	140	188
	NSH	9	100	144	172	112	152	200
Schedule			80	80	104	80	80	104
GetTaskID			36	36	36	36	36	36
GetTaskState			72	72	72	88	88	88
EnableAllInterrupts			28	28	28	28	28	28
DisableAllInterrupts			28	28	28	28	28	28
ResumeAllInterrupts			44	44	44	44	44	44
SuspendAllInterrupts			40	40	40	40	40	40
ResumeOSInterrupts			44	44	44	44	44	44
SuspendOSInterrupts			48	48	48	48	48	48
GetResource	Task	7	32	32	36	32	32	36
	Combined	6	56	56	56	56	56	56
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	64	64	64	64	64	64
	Combined	6	100	100	100	100	100	100
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	104	104	172
	NS		n/a	n/a	n/a	80	80	148
	NS1i	10	n/a	n/a	n/a	44	n/a	n/a
	KL	2	n/a	n/a	n/a	68	68	136
	KL1i	2, 10	n/a	n/a	n/a	28	n/a	n/a
ClearEvent			n/a	n/a	n/a	44	44	44
GetEvent			n/a	n/a	n/a	18	18	18
WaitEvent	<default>		n/a	n/a	n/a	180	180	316
	fp	11	n/a	n/a	n/a	204	204	364
	1i	10	n/a	n/a	n/a	28	n/a	n/a
GetAlarmBase			38	38	38	38	38	38
GetAlarm			74	74	74	74	74	74
SetRelAlarm			100	100	100	100	100	100
SetAbsAlarm			110	110	110	110	110	110
CancelAlarm			62	62	62	62	62	62
InitCounter			44	44	44	44	44	44

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetCounterValue			54	54	54	54	54	54
osek_tick_alarm	<default>		60	60	60	60	60	60
	KL	2	40	40	40	40	40	40
osek_incr_counter			34	34	34	34	34	34
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			128	128	128	128	128	128
ShutdownOS	NoHook	12	24	24	24	24	24	24
	Hook	13	32	32	32	32	32	32
InitCOM			12	12	12	12	12	12
CloseCOM			12	12	12	12	12	12
StartCOM			30	30	30	30	30	30
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	50	50	50	134	134	134
	CCCB	15	134	134	134	134	134	134
GetMessageResource			48	48	48	48	48	48
ReleaseMessageResource			48	48	48	48	48	48
GetMessageStatus			46	46	46	46	46	46
SendMessage	SW CCCA	1, 14	70	70	70	172	172	172
	SW CCCB	1, 15	160	160	160	172	172	172
	NS CCCA	14	70	70	70	172	172	172
	NS CCCB	15	160	160	160	172	172	172
	KL CCCA	2, 14	54	54	54	156	156	156
	KL CCCB	2, 15	144	144	144	156	156	156
main_dispatch	NoHook	12	96	96	128	96	96	128
	Hook	13	132	132	164	132	132	164
sub_dispatch	B1LF	19	40	40	40	40	40	40
	B1HI	20	80	80	80	80	80	80
	B1HF	21	88	88	88	88	88	88
	B2LI	22	n/a	64	84	n/a	64	84
	B2LF	23	n/a	72	92	n/a	72	92
	B2HI	24	n/a	112	160	n/a	112	160
	B2HF	25	n/a	120	168	n/a	120	168
	E1HI	26	n/a	n/a	n/a	252	252	304
	E1HF	27	n/a	n/a	n/a	260	260	312
	E2HI	28	n/a	n/a	n/a	n/a	n/a	304

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	E2HF	29	n/a	n/a	n/a	n/a	n/a	312
ErrorHook support		16	44	44	44	44	44	44
	ServiceID	17	52	52	52	52	52	52
	Parameters	18	60	60	60	60	60	60
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	68	112	148	76	132	172
	NS		44	88	124	52	108	152
	KL	2	28	76	108	36	96	136
ChainTaskset	SWL	1, 8	52	96	128	52	108	144
	SWH	1, 9	88	132	168	88	140	184
	NSL	8	52	96	128	52	108	144
	NSH	9	76	120	156	76	128	172
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			38	38	38	38	38	38
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			38	38	38	38	38	38
TestSubTaskset			46	46	46	46	46	46
TestEquivalentTaskset			46	46	46	46	46	46
TickSchedule	SW	1	120	120	120	120	120	120
	NS		96	92	92	92	92	92
	KL	2	84	80	80	80	80	80
AdvanceSchedule	SW	1	112	108	108	108	108	108
	NS		92	80	80	80	80	80
	KL	2	76	68	68	68	68	68
StartSchedule			66	66	66	66	66	66
StopSchedule			50	50	50	50	50	50
GetScheduleStatus			76	76	76	76	76	76
GetScheduleValue			56	56	56	56	56	56
GetScheduleNext			18	18	18	18	18	18
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			14	14	14	14	14	14
GetArrivalpointTasksetRef			14	14	14	14	14	14
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			14	14	14	14	14	14

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestArrivalpointWritable			40	40	40	40	40	40
GetExecutionTime			12	12	12	12	12	12
GetLargestExecutionTime			14	14	14	14	14	14
ResetLargestExecutionTime			12	12	12	12	12	12
GetStackOffset			36	36	36	36	36	36

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	100	132	164	112	144	196
	NS		80	112	144	92	124	176
	KL	2	60	88	120	72	100	156
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	36	36	36	36	36	36
ChainTask	SWL	1, 8	80	132	160	100	140	188
	SWH	1, 9	112	156	184	124	164	212
	NSL	8	80	132	160	100	140	188
	NSH	9	100	144	172	112	152	200
Schedule			100	100	124	100	100	124
GetTaskID			36	36	36	36	36	36
GetTaskState			72	72	72	88	88	88
EnableAllInterrupts			28	28	28	28	28	28
DisableAllInterrupts			28	28	28	28	28	28
ResumeAllInterrupts			44	44	44	44	44	44
SuspendAllInterrupts			40	40	40	40	40	40
ResumeOSInterrupts			44	44	44	44	44	44
SuspendOSInterrupts			48	48	48	48	48	48
GetResource	Task	7	32	32	36	32	32	36
	Combined	6	56	56	56	56	56	56
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	84	84	84	84	84	84
	Combined	6	136	136	136	136	136	136

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	104	104	172
	NS		n/a	n/a	n/a	80	80	148
	NS1i	10	n/a	n/a	n/a	44	n/a	n/a
	KL	2	n/a	n/a	n/a	68	68	136
	KL1i	2, 10	n/a	n/a	n/a	28	n/a	n/a
ClearEvent			n/a	n/a	n/a	44	44	44
GetEvent			n/a	n/a	n/a	18	18	18
WaitEvent	<default>		n/a	n/a	n/a	232	232	368
	fp	11	n/a	n/a	n/a	252	252	416
	1i	10	n/a	n/a	n/a	80	n/a	n/a
GetAlarmBase			38	38	38	38	38	38
GetAlarm			74	74	74	74	74	74
SetRelAlarm			100	100	100	100	100	100
SetAbsAlarm			110	110	110	110	110	110
CancelAlarm			62	62	62	62	62	62
InitCounter			44	44	44	44	44	44
GetCounterValue			54	54	54	54	54	54
osek_tick_alarm	<default>		60	60	60	60	60	60
	KL	2	40	40	40	40	40	40
osek_incr_counter			34	34	34	34	34	34
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			172	172	172	172	172	172
ShutdownOS	NoHook	12	24	24	24	24	24	24
	Hook	13	32	32	32	32	32	32
InitCOM			12	12	12	12	12	12
CloseCOM			12	12	12	12	12	12
StartCOM			30	30	30	30	30	30
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	50	50	50	134	134	134
	CCCB	15	134	134	134	134	134	134
GetMessageResource			48	48	48	48	48	48
ReleaseMessageResource			48	48	48	48	48	48
GetMessageStatus			46	46	46	46	46	46
SendMessage	SW CCCA	1, 14	70	70	70	172	172	172

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
	SW CCCB	1, 15	160	160	160	172	172	172
	NS CCCA	14	70	70	70	172	172	172
	NS CCCB	15	160	160	160	172	172	172
	KL CCCA	2, 14	54	54	54	156	156	156
	KL CCCB	2, 15	144	144	144	156	156	156
main_dispatch	NoHook	12	156	156	188	156	156	188
	Hook	13	196	196	228	196	196	228
sub_dispatch	B1LF	19	28	28	28	28	28	28
	B1HI	20	88	88	88	88	88	88
	B1HF	21	96	96	96	96	96	96
	B2LI	22	n/a	56	76	n/a	56	76
	B2LF	23	n/a	64	84	n/a	64	84
	B2HI	24	n/a	112	160	n/a	112	160
	B2HF	25	n/a	120	168	n/a	120	168
	E1HI	26	n/a	n/a	n/a	284	284	332
	E1HF	27	n/a	n/a	n/a	292	292	340
	E2HI	28	n/a	n/a	n/a	n/a	n/a	332
	E2HF	29	n/a	n/a	n/a	n/a	n/a	340
ErrorHook support		16	44	44	44	44	44	44
	ServiceID	17	52	52	52	52	52	52
	Parameters	18	60	60	60	60	60	60
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	72	72	72	72	72	72
Timing_termination		4	76	76	76	76	76	76
ActivateTaskset	SW	1	68	112	148	76	132	172
	NS		44	88	124	52	108	152
	KL	2	28	76	108	36	96	136
ChainTaskset	SWL	1, 8	52	96	128	52	108	144
	SWH	1, 9	88	132	168	88	140	184
	NSL	8	52	96	128	52	108	144
	NSH	9	76	120	156	76	128	172
GetTasksetRef			16	16	16	16	16	16
MergeTaskset			38	38	38	38	38	38
AssignTaskset			16	16	16	16	16	16
RemoveTaskset			38	38	38	38	38	38
TestSubTaskset			46	46	46	46	46	46
TestEquivalentTaskset			46	46	46	46	46	46

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TickSchedule	SW	1	120	120	120	120	120	120
	NS		96	92	92	92	92	92
	KL	2	84	80	80	80	80	80
AdvanceSchedule	SW	1	112	108	108	108	108	108
	NS		92	80	80	80	80	80
	KL	2	76	68	68	68	68	68
StartSchedule			66	66	66	66	66	66
StopSchedule			50	50	50	50	50	50
GetScheduleStatus			76	76	76	76	76	76
GetScheduleValue			56	56	56	56	56	56
GetScheduleNext			18	18	18	18	18	18
SetScheduleNext			16	16	16	16	16	16
GetArrivalpointDelay			16	16	16	16	16	16
SetArrivalpointDelay			14	14	14	14	14	14
GetArrivalpointTasksetRef			14	14	14	14	14	14
GetArrivalpointNext			16	16	16	16	16	16
SetArrivalpointNext			14	14	14	14	14	14
TestArrivalpointWritable			40	40	40	40	40	40
GetExecutionTime			88	88	88	88	88	88
GetLargestExecutionTime			20	20	20	20	20	20
ResetLargestExecutionTime			20	20	20	20	20	20
GetStackOffset			36	36	36	36	36	36

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	180	216	244	192	224	280
	NS		212	248	276	224	256	312
	KL	2	132	168	196	148	180	236
TerminateTask	LExt	3	108	108	108	108	108	108
	H	5	128	128	128	128	128	128
ChainTask	SWL	1, 8	204	256	284	224	268	316

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	SWH	1, 9	240	288	316	256	296	348
	NSL	8	248	300	328	268	312	360
	NSH	9	276	320	348	288	332	380
Schedule			180	180	204	180	180	204
GetTaskID			52	52	52	52	52	52
GetTaskState			180	180	180	184	184	184
EnableAllInterrupts			44	44	44	44	44	44
DisableAllInterrupts			40	40	40	40	40	40
ResumeAllInterrupts			76	76	76	76	76	76
SuspendAllInterrupts			52	52	52	52	52	52
ResumeOSInterrupts			76	76	76	76	76	76
SuspendOSInterrupts			64	64	64	64	64	64
GetResource	Task	7	240	240	216	240	240	216
	Combined	6	220	220	220	220	220	220
	CLEx	3	208	208	208	208	208	208
ReleaseResource	Task	7	244	244	244	244	244	244
	Combined	6	288	288	288	288	288	288
	CLEx	3	200	200	200	200	200	200
SetEvent	SW	1	n/a	n/a	n/a	224	224	296
	NS		n/a	n/a	n/a	256	256	324
	NS1i	10	n/a	n/a	n/a	160	n/a	n/a
	KL	2	n/a	n/a	n/a	180	180	248
	KL1i	2, 10	n/a	n/a	n/a	132	n/a	n/a
ClearEvent			n/a	n/a	n/a	104	104	104
GetEvent			n/a	n/a	n/a	140	140	140
WaitEvent	<default>		n/a	n/a	n/a	316	316	444
	fp	11	n/a	n/a	n/a	340	340	500
	1i	10	n/a	n/a	n/a	168	n/a	n/a
GetAlarmBase			116	116	116	116	116	116
GetAlarm			124	124	124	124	124	124
SetRelAlarm			176	176	176	176	176	176
SetAbsAlarm			184	184	184	184	184	184
CancelAlarm			112	112	112	112	112	112
InitCounter			156	156	156	156	156	156
GetCounterValue			132	132	132	132	132	132
osek_tick_alarm	<default>		78	78	78	78	78	78
	KL	2	40	40	40	40	40	40

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
osek_incr_counter			34	34	34	34	34	34
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			188	188	188	188	188	188
ShutdownOS	NoHook	12	36	36	36	36	36	36
	Hook	13	44	44	44	44	44	44
InitCOM			12	12	12	12	12	12
CloseCOM			12	12	12	12	12	12
StartCOM			44	44	44	44	44	44
StopCOM			52	52	52	52	52	52
ReadFlag			32	32	32	32	32	32
ResetFlag			36	36	36	36	36	36
ReceiveMessage	CCCA	14	116	116	116	200	200	200
	CCCB	15	200	200	200	200	200	200
GetMessageResource			88	88	88	88	88	88
ReleaseMessageResource			88	88	88	88	88	88
GetMessageStatus			88	88	88	88	88	88
SendMessage	SW CCCA	1, 14	140	140	140	244	244	244
	SW CCCB	1, 15	232	232	232	244	244	244
	NS CCCA	14	140	140	140	244	244	244
	NS CCCB	15	232	232	232	244	244	244
	KL CCCA	2, 14	120	120	120	224	224	224
	KL CCCB	2, 15	212	212	212	224	224	224
main_dispatch	NoHook	12	156	156	188	156	156	188
	Hook	13	196	196	228	196	196	228
sub_dispatch	B1LF	19	28	28	28	28	28	28
	B1HI	20	88	88	88	88	88	88
	B1HF	21	96	96	96	96	96	96
	B2LI	22	n/a	56	76	n/a	56	76
	B2LF	23	n/a	64	84	n/a	64	84
	B2HI	24	n/a	112	160	n/a	112	160
	B2HF	25	n/a	120	168	n/a	120	168
	E1HI	26	n/a	n/a	n/a	284	284	332
	E1HF	27	n/a	n/a	n/a	292	292	340
	E2HI	28	n/a	n/a	n/a	n/a	n/a	332
	E2HF	29	n/a	n/a	n/a	n/a	n/a	340
ErrorHook support		16	88	88	88	88	88	88
	ServiceID	17	100	100	100	100	100	100

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
	Parameters	18	124	124	124	124	124	124
validity_checks		3	30	30	30	30	30	30
Timing_dispatch		4	72	72	72	72	72	72
Timing_termination		4	76	76	76	76	76	76
ActivateTaskset	SW	1	224	296	320	268	328	364
	NS		252	320	344	292	352	388
	KL	2	172	240	268	212	268	312
ChainTaskset	SWL	1, 8	268	348	368	304	368	412
	SWH	1, 9	312	392	420	352	416	460
	NSL	8	312	392	420	356	412	456
	NSH	9	344	424	452	392	448	492
GetTasksetRef			108	108	108	108	108	108
MergeTaskset			176	176	176	176	176	176
AssignTaskset			132	132	132	132	132	132
RemoveTaskset			176	176	176	176	176	176
TestSubTaskset			184	184	184	184	184	184
TestEquivalentTaskset			184	184	184	184	184	184
TickSchedule	SW	1	252	200	200	200	200	200
	NS		280	244	244	244	244	244
	KL	2	216	160	160	160	160	160
AdvanceSchedule	SW	1	260	204	204	204	204	204
	NS		288	248	248	248	248	248
	KL	2	224	164	164	164	164	164
StartSchedule			172	172	172	172	172	172
StopSchedule			128	128	128	128	128	128
GetScheduleStatus			156	156	156	156	156	156
GetScheduleValue			128	128	128	128	128	128
GetScheduleNext			76	76	76	76	76	76
SetScheduleNext			140	140	140	140	140	140
GetArrivalpointDelay			104	104	104	104	104	104
SetArrivalpointDelay			116	116	116	116	116	116
GetArrivalpointTasksetRef			100	100	100	100	100	100
GetArrivalpointNext			104	104	104	104	104	104
SetArrivalpointNext			136	136	136	136	136	136
TestArrivalpointWritable			112	112	112	112	112	112
GetExecutionTime			116	116	116	116	116	116
GetLargestExecutionTime			96	96	96	96	96	96

Configuration	Application Uses					
Events	No			Yes		
Shared Task Priorities	No		Yes	No		Yes
Multiple Task Activations	No	Yes		No	Yes	
ResetLargestExecutionTime	92	92	92	92	92	92
GetStackOffset	36	36	36	36	36	36

Notes

Number	Note
1	Linked only if upward activations are allowed
2	Linked only if API is called within ISR
3	Present only in Extended OS status
4	Present only in Timing or Extended OS status
5	Linked only if there are heavyweight tasks in the system
6	Linked only if Resource is used by both tasks and ISRs
7	Linked only if Resource is used only by tasks
8	Linked only if Chaining task is Lightweight
9	Linked only if Chaining task is Heavyweight
10	Linked only if Idle task is the only extended task in the system
11	Linked only if calling Extended task uses floating-point
12	Linked only if neither Pre- nor Post-TaskHook is used
13	Linked only if Pre- or Post-TaskHook is used
14	Linked only if there are no flags, message queues, or message resources in the system, and COM status is not requested.
15	Linked only if there are any flags, message queues, or message resources in the system, or COM status is requested.
16	Linked only if `USEGETSERVICEID = FALSE` 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

Number	Note
27	Linked only for extended, unique priority, floating-point tasks
28	Linked only for extended, shared priority, integer tasks
29	Linked only for extended, shared priority, floating-point tasks
30	Implemented as a macro, so no code is linked
31	Not required on some targets

4.2.4 Reserved Hardware Resources

Timer units, interrupts, traps and other hardware resources are not reserved by RTA-OSEK.

4.3 Performance

The collection of performance data for the TMS470/ADS port of the RTA-OSEK Component was achieved using a timer running two times slower than the CPU clock speed. The figures in this section, therefore, have an uncertainty level of up to two CPU cycles. The actual times are between 0 and two cycles shorter than those reported in the remainder of this section.

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	120	176	210	128	162	218
	NS	110	158	192	116	150	200
	KL	66	110	142	74	106	146
TerminateTask	LExt	0	0	0	0	0	0
	H	210	212	224	208	214	224
ChainTask	SWL	338	380	448	412	440	530

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	SWH	422	470	542	500	538	624
	NSL	336	380	444	410	440	528
	NSH	410	456	526	486	526	608
Schedule	SW	118	118	130	118	116	132
GetTaskID		42	42	40	42	38	40
GetTaskState		116	116	116	122	122	124
EnableAllInterrupts		48	50	54	52	50	48
DisableAllInterrupts		74	72	78	74	72	70
ResumeAllInterrupts		62	60	64	62	60	58
SuspendAllInterrupts		82	82	86	84	80	80
ResumeOSInterrupts		60	60	62	62	60	60
SuspendOSInterrupts		82	80	84	84	82	82
GetResource	Task	50	48	50	48	44	46
	Combined	68	70	68	68	64	66
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	100	102	100	102	98	100
	Combined	142	142	140	142	144	144
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	136	132	134
	NS	n/a	n/a	n/a	132	124	136
	KL	n/a	n/a	n/a	84	80	86
ClearEvent		n/a	n/a	n/a	90	84	86
GetEvent		n/a	n/a	n/a	36	34	34
WaitEvent	<default>	n/a	n/a	n/a	554	562	630
	fp	n/a	n/a	n/a	566	564	640
GetAlarmBase		98	98	98	98	96	98
GetAlarm		118	118	118	118	112	116
SetRelAlarm		126	126	124	126	122	122
SetAbsAlarm		126	126	126	128	126	126
CancelAlarm		100	100	100	100	98	100
InitCounter		94	96	96	96	94	94
GetCounterValue		102	102	104	102	102	100
osek_tick_alarm	<default>	100	100	100	100	96	100
	KL	40	40	38	40	38	36
osek_incr_counter		18	18	18	18	16	16
GetActiveApplicationMode		20	18	20	18	18	18

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
StartOS		1294	1294	1294	1294	1294	1292
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	64	64	68	66	64	62
InitCOM		20	18	20	20	18	18
CloseCOM		16	18	18	16	14	16
StartCOM		56	52	50	154	158	158
StopCOM		32	32	34	32	32	32
ReadFlag		n/a	n/a	n/a	32	30	30
ResetFlag		n/a	n/a	n/a	24	22	22
ReceiveMessage		86	88	88	318	318	320
GetMessageResource		n/a	n/a	n/a	106	104	104
ReleaseMessageResource		n/a	n/a	n/a	156	152	156
GetMessageStatus		n/a	n/a	n/a	58	56	56
SendMessage	SW	222	276	310	450	480	538
	NS	204	252	288	430	464	516
	KL	118	162	194	350	382	424
ActivateTaskset	SW	102	570	646	112	606	662
	NS	78	580	594	86	552	610
	KL	30	504	548	36	540	562
	SW2	102	572	646	112	606	662
	NS2	78	582	594	86	552	610
	KL2	30	504	546	36	540	562
ChainTaskset	SWL	318	788	924	386	890	980
	SWH	412	912	1018	480	944	1066
	NSL	314	782	922	382	888	976
	NSH	404	904	950	472	938	1060
GetTasksetRef		34	34	36	34	32	34
MergeTaskset		96	96	96	98	92	96
AssignTaskset		26	28	26	28	24	26
RemoveTaskset		86	86	86	88	82	86
TestSubTaskset		96	96	96	96	92	96
TestEquivalentTaskset		92	92	92	92	88	92
TickSchedule	SW	164	660	704	192	704	724
	NS	140	632	674	166	674	696
	KL	94	588	630	120	634	652
	SW2	164	660	702	194	696	718

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NS2	140	632	676	164	666	690
	KL2	94	588	632	120	626	646
AdvanceSchedule	SW	150	638	680	170	680	702
	NS	124	608	652	142	656	676
	KL	78	572	616	104	618	636
	SW2	150	638	680	170	674	696
	NS2	124	608	652	142	650	672
	KL2	78	574	616	104	610	630
StartSchedule		126	126	126	126	118	120
StopSchedule		108	108	108	108	108	110
GetScheduleStatus		118	118	120	120	116	116
GetScheduleValue		118	118	116	118	114	116
GetScheduleNext		38	38	38	38	36	38
SetScheduleNext		38	38	40	38	38	36
GetArrivalpointDelay		36	36	36	36	34	34
SetArrivalpointDelay		36	34	36	34	34	34
GetArrivalpointTasksetRef		32	32	32	32	32	30
GetArrivalpointNext		36	36	34	36	32	34
SetArrivalpointNext		36	36	36	36	36	34
TestArrivalpointWritable		48	50	50	50	48	46
GetExecutionTime		20	20	20	20	18	18
GetLargestExecutionTime		26	26	26	26	24	24
ResetLargestExecutionTime		18	18	18	18	16	16
GetStackOffset		46	46	46	46	44	44

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	122	174	210	132	164	214
	NS	106	160	192	116	152	206
	KL	62	110	142	70	108	152

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
TerminateTask	LExt	0	0	0	0	0	0
	H	452	460	466	454	456	466
ChainTask	SWL	610	662	720	694	718	810
	SWH	698	746	810	780	814	896
	NSL	608	660	716	694	720	810
	NSH	682	732	794	766	800	884
Schedule	SW	116	118	134	116	116	134
GetTaskID		38	40	40	40	42	40
GetTaskState		114	118	114	124	122	120
EnableAllInterrupts		48	50	50	50	50	50
DisableAllInterrupts		74	76	76	70	70	70
ResumeAllInterrupts		60	62	62	60	60	60
SuspendAllInterrupts		82	84	84	82	82	82
ResumeOSInterrupts		60	62	62	62	62	62
SuspendOSInterrupts		82	84	84	82	82	82
GetResource	Task	46	50	48	48	48	48
	Combined	68	70	70	68	68	68
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	104	106	104	104	104	104
	Combined	148	150	144	150	150	146
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	134	134	140
	NS	n/a	n/a	n/a	126	126	136
	KL	n/a	n/a	n/a	82	82	86
ClearEvent		n/a	n/a	n/a	90	90	90
GetEvent		n/a	n/a	n/a	36	36	36
WaitEvent	<default>	n/a	n/a	n/a	804	804	856
	fp	n/a	n/a	n/a	814	814	866
GetAlarmBase		98	100	98	98	98	98
GetAlarm		116	118	118	116	114	118
SetRelAlarm		124	124	124	124	122	126
SetAbsAlarm		126	128	126	126	128	126
CancelAlarm		100	102	100	100	100	100
InitCounter		96	98	98	96	96	98
GetCounterValue		104	104	104	104	102	104
osek_tick_alarm	<default>	100	102	100	100	98	100

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	KL	38	38	40	38	40	40
osek_incr_counter		16	18	18	18	18	18
GetActiveApplicationMode		16	20	18	18	18	18
StartOS		3116	3118	3116	3118	3118	3118
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	64	64	68	66	66	66
InitCOM		16	20	18	18	18	20
CloseCOM		14	18	16	18	18	16
StartCOM		50	56	52	162	162	154
StopCOM		30	32	34	34	34	32
ReadFlag		n/a	n/a	n/a	32	32	32
ResetFlag		n/a	n/a	n/a	24	24	24
ReceiveMessage		86	86	88	320	320	316
GetMessageResource		n/a	n/a	n/a	108	108	108
ReleaseMessageResource		n/a	n/a	n/a	158	158	160
GetMessageStatus		n/a	n/a	n/a	58	58	58
SendMessage	SW	222	274	310	448	482	536
	NS	200	254	288	432	466	520
	KL	116	166	194	346	384	430
ActivateTaskset	SW	106	572	646	110	608	666
	NS	80	644	594	82	554	670
	KL	28	504	546	36	542	624
	SW2	106	572	646	110	608	664
	NS2	80	644	594	82	554	672
	KL2	28	504	548	36	542	624
ChainTaskset	SWL	594	1066	1196	668	1170	1318
	SWH	682	1252	1286	756	1220	1340
	NSL	590	1064	1194	664	1168	1314
	NSH	678	1244	1218	750	1214	1332
GetTasksetRef		32	34	36	36	34	36
MergeTaskset		94	98	96	96	94	98
AssignTaskset		24	28	26	28	26	28
RemoveTaskset		84	88	86	86	84	88
TestSubTaskset		94	96	96	96	94	96
TestEquivalentTaskset		90	92	92	92	90	92
TickSchedule	SW	164	662	704	192	706	786

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
	NS	138	632	674	164	678	758
	KL	92	588	634	120	634	716
	SW2	164	660	702	192	696	780
	NS2	138	634	676	164	670	752
	KL2	92	588	632	120	626	710
AdvanceSchedule	SW	152	640	682	170	684	764
	NS	124	616	652	146	658	736
	KL	76	574	616	106	620	700
	SW2	152	640	682	170	676	758
	NS2	124	616	652	146	652	730
	KL2	76	574	616	108	612	694
StartSchedule		126	128	120	126	124	122
StopSchedule		108	110	110	108	106	112
GetScheduleStatus		118	120	118	118	118	118
GetScheduleValue		116	118	116	116	116	118
GetScheduleNext		36	38	40	38	38	38
SetScheduleNext		38	40	38	40	38	40
GetArrivalpointDelay		34	36	36	36	36	36
SetArrivalpointDelay		34	36	34	36	36	36
GetArrivalpointTasksetRef		30	34	32	34	32	32
GetArrivalpointNext		32	34	36	34	36	34
SetArrivalpointNext		34	38	36	38	36	36
TestArrivalpointWritable		48	50	50	50	48	50
GetExecutionTime		144	150	144	144	144	148
GetLargestExecutionTime		32	34	34	34	34	34
ResetLargestExecutionTime		24	28	26	28	26	28
GetStackOffset		44	46	46	46	46	46

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	330	374	412	336	372	418
	NS	356	404	436	366	400	450
	KL	268	314	344	280	312	358
TerminateTask	LExt	502	504	514	510	508	518
	H	564	564	572	566	568	574
ChainTask	SWL	874	920	984	960	986	1076
	SWH	966	1010	1072	1046	1090	1162
	NSL	912	958	1020	996	1022	1112
	NSH	994	1036	1102	1070	1106	1186
Schedule	SW	176	176	192	180	180	196
GetTaskID		48	50	48	52	50	50
GetTaskState		332	330	332	332	332	334
EnableAllInterrupts		56	58	58	60	60	60
DisableAllInterrupts		84	84	86	86	88	84
ResumeAllInterrupts		80	78	80	80	80	80
SuspendAllInterrupts		90	90	92	92	90	90
ResumeOSInterrupts		78	78	80	82	80	80
SuspendOSInterrupts		90	88	92	94	92	92
GetResource	Task	496	496	310	542	568	356
	Combined	258	258	258	304	300	302
	CLEx	306	306	308	350	350	348
ReleaseResource	Task	296	296	296	342	344	340
	Combined	318	318	316	360	370	362
	CLEx	282	282	282	328	326	326
SetEvent	SW	n/a	n/a	n/a	362	360	358
	NS	n/a	n/a	n/a	378	378	376
	KL	n/a	n/a	n/a	310	310	312
ClearEvent		n/a	n/a	n/a	146	142	146
GetEvent		n/a	n/a	n/a	270	270	270
WaitEvent	<default>	n/a	n/a	n/a	896	900	942
	fp	n/a	n/a	n/a	904	904	950
GetAlarmBase		250	250	250	254	254	252
GetAlarm		264	264	266	270	268	268

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
SetRelAlarm		296	296	296	302	302	300
SetAbsAlarm		296	296	296	302	300	300
CancelAlarm		242	242	242	240	240	238
InitCounter		378	378	378	382	380	380
GetCounterValue		244	244	242	246	244	246
osek_tick_alarm	<default>	124	124	124	130	128	128
	KL	38	38	36	38	38	40
osek_incr_counter		18	18	16	18	20	20
GetActiveApplicationMode		18	16	16	18	18	18
StartOS		3242	3242	3242	3242	3242	3244
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	70	74	74	78	72	76
InitCOM		16	16	14	18	18	18
CloseCOM		18	18	18	20	20	20
StartCOM		76	76	76	178	176	176
StopCOM		48	48	46	50	50	50
ReadFlag		n/a	n/a	n/a	54	54	54
ResetFlag		n/a	n/a	n/a	46	46	46
ReceiveMessage		206	200	200	436	434	434
GetMessageResource		n/a	n/a	n/a	494	486	494
ReleaseMessageResource		n/a	n/a	n/a	488	492	488
GetMessageStatus		n/a	n/a	n/a	152	152	152
SendMessage	SW	544	588	626	768	804	850
	NS	562	610	640	796	830	880
	KL	432	478	508	668	700	744
ActivateTaskset	SW	656	1190	1282	688	1198	1318
	NS	670	1274	1216	710	1278	1304
	KL	580	1126	1190	622	1106	1256
	SW2	656	1190	1284	690	1198	1318
	NS2	670	1274	1216	710	1278	1304
	KL2	580	1126	1188	622	1106	1256
ChainTaskset	SWL	1192	1828	1866	1458	1800	2086
	SWH	1284	1918	1918	1516	1950	2180
	NSL	1260	1800	1928	1530	1898	2056
	NSH	1344	1884	1954	1582	1920	2082
GetTasksetRef		242	240	242	244	244	242

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
MergeTaskset		186	184	186	188	186	186
AssignTaskset		104	106	104	106	106	108
RemoveTaskset		174	172	174	176	176	174
TestSubTaskset		194	192	194	196	194	194
TestEquivalentTaskset		190	188	190	192	190	190
TickSchedule	SW	252	1330	1392	826	1324	1474
	NS	278	1354	1416	848	1350	1496
	KL	188	1266	1330	760	1260	1408
	SW2	252	1330	1392	824	1306	1460
	NS2	276	1352	1416	848	1334	1484
	KL2	188	1266	1330	762	1242	1396
AdvanceSchedule	SW	238	1322	1386	816	1314	1462
	NS	256	1346	1408	840	1340	1488
	KL	176	1258	1320	756	1254	1402
	SW2	236	1324	1386	816	1298	1450
	NS2	256	1346	1410	840	1324	1474
	KL2	176	1258	1320	756	1238	1390
StartSchedule		198	198	200	202	200	202
StopSchedule		162	162	162	166	170	164
GetScheduleStatus		176	176	176	178	176	178
GetScheduleValue		172	172	172	174	176	174
GetScheduleNext		80	80	82	84	82	84
SetScheduleNext		122	122	120	122	126	122
GetArrivalpointDelay		96	98	98	100	98	100
SetArrivalpointDelay		106	106	108	110	108	110
GetArrivalpointTasksetRef		82	82	82	84	84	84
GetArrivalpointNext		84	88	86	88	86	88
SetArrivalpointNext		120	118	120	122	122	122
TestArrivalpointWritable		94	92	92	94	96	94
GetExecutionTime		174	174	174	176	176	176
GetLargestExecutionTime		224	224	224	228	226	226
ResetLargestExecutionTime		214	218	218	220	216	220
GetStackOffset		44	44	44	46	46	46

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	78	78	78	78	78	78
	Cat 2	108	106	108	108	108	108

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	78	78	78	78	78	78
	Cat 2	244	244	242	242	242	244

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	78	78	78	78	78	78
	Cat 2	248	242	242	242	242	248

4.3.4 Task Switching Times

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

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

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

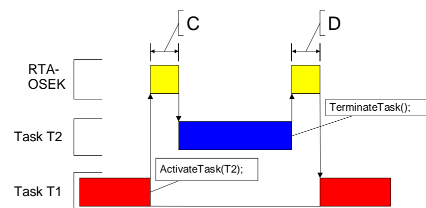


Figure 1: Task Activates a Higher Priority Task which Terminates Normally

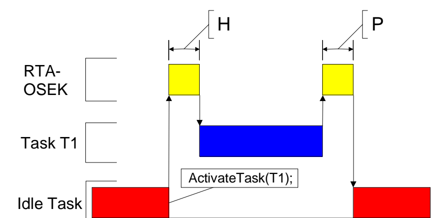


Figure 3: Task Activation from Idle Task

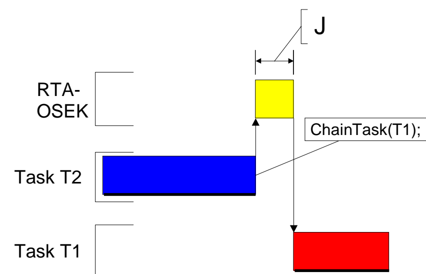


Figure 2: Task Chaining

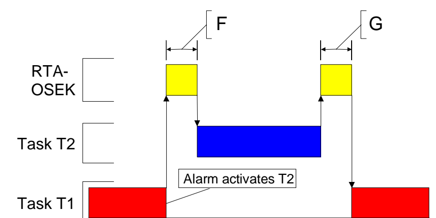


Figure 4: Task Activation from an Alarm

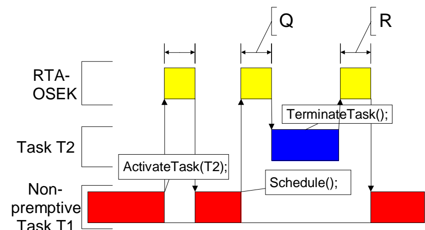


Figure 5: Non-Premptive Task Calls Schedule()

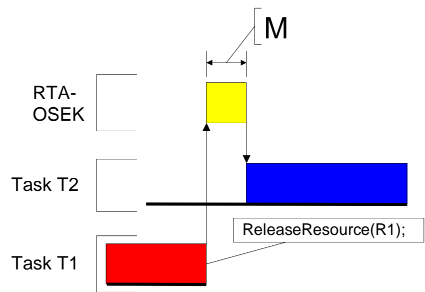


Figure 6: Blocked Task Activated by ReleaseResource()

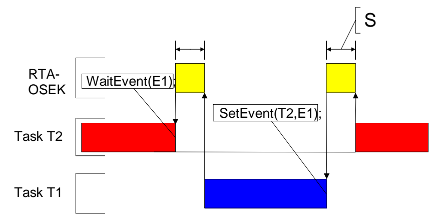


Figure 7: Waiting Task Activated by SetEvent()

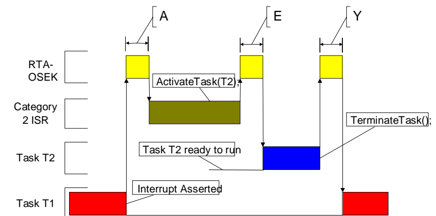


Figure 8: Category 2 ISR Activates a Higher Priority Task

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Normal termination	Light, Basic	120	170	202	118	168	200
Figure 1: D	Heavy, Basic/Extended	214	262	288	272	272	296
ChainTask	Light, Basic	256	322	392	262	320	404
Figure 2: J	Heavy, Basic/Extended	588	704	804	656	726	832
Pre-emption	Light, Basic	200	278	352	206	268	364
Figure 1: C	Heavy, Basic/Extended	284	340	412	360	396	494
From idle task	Light, Basic	206	282	356	212	274	370
Figure 3: H	Heavy, Basic/Extended	290	344	416	366	402	500
Triggered by alarm	Light, Basic	310	386	460	316	376	474
Figure 4: F	Heavy, Basic/Extended	394	446	518	470	504	604
Schedule	Light, Basic	192	216	272	194	212	278
Figure 5: Q	Heavy, Basic/Extended	278	278	332	346	342	408
Release resource	Light, Basic	200	226	268	200	220	268
Figure 6: M	Heavy, Basic/Extended	286	288	328	352	350	400
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	650	644	788
From category 2 ISR	Light, Basic	154	180	220	154	176	224
Figure 8: E	Heavy, Basic/Extended	240	242	280	308	308	356

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	366	406	436	366	402	436
Figure 1: D	Heavy, Basic/Extended	456	484	504	494	494	524
ChainTask	Light, Basic	538	598	658	550	594	682
Figure 2: J	Heavy, Basic/Extended	1118	1196	1286	1164	1216	1320
Pre-emption	Light, Basic	352	412	490	360	408	500
Figure 1: C	Heavy, Basic/Extended	426	480	554	508	542	632
From idle task	Light, Basic	356	416	496	366	414	506
Figure 3: H	Heavy, Basic/Extended	432	484	560	512	548	638
Triggered by alarm	Light, Basic	462	522	600	468	516	608
Figure 4: F	Heavy, Basic/Extended	536	588	664	616	650	740
Schedule	Light, Basic	338	350	412	340	352	414
Figure 5: Q	Heavy, Basic/Extended	414	418	478	488	488	548
Release resource	Light, Basic	350	360	404	352	360	404
Figure 6: M	Heavy, Basic/Extended	424	428	470	498	498	540
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	776	776	890
From category 2 ISR	Light, Basic	582	594	632	582	592	636
Figure 8: E	Heavy, Basic/Extended	656	662	698	730	728	772

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	498	534	566	504	538	570
Figure 1: D	Heavy, Basic/Extended	568	588	612	608	602	634
ChainTask	Light, Basic	804	856	926	818	858	946
Figure 2: J	Heavy, Basic/Extended	1500	1568	1658	1544	1588	1694
Pre-emption	Light, Basic	554	606	688	560	606	694
Figure 1: C	Heavy, Basic/Extended	626	672	750	704	734	828
From idle task	Light, Basic	558	612	694	566	612	698
Figure 3: H	Heavy, Basic/Extended	632	678	756	710	740	834
Triggered by alarm	Light, Basic	686	740	820	698	744	830
Figure 4: F	Heavy, Basic/Extended	760	806	884	842	870	964
Schedule	Light, Basic	392	402	460	394	402	464
Figure 5: Q	Heavy, Basic/Extended	466	468	524	538	534	596
Release resource	Light, Basic	526	534	578	570	580	624
Figure 6: M	Heavy, Basic/Extended	600	600	642	714	716	756
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	994	990	1098
From category 2 ISR	Light, Basic	616	626	670	616	626	672
Figure 8: E	Heavy, Basic/Extended	690	692	734	762	760	804

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	40	40	48	40	40	48
BCC1 lightweight, floating-point	48	48	56	48	48	56
BCC1 heavyweight, integer	88	88	96	88	88	96
BCC1 heavyweight, floating-point	88	88	96	88	88	96
BCC2 lightweight, integer	n/a	48	56	n/a	48	56
BCC2 lightweight, floating-point	n/a	48	56	n/a	48	56
BCC2 heavyweight, integer	n/a	96	96	n/a	96	96
BCC2 heavyweight, floating-point	n/a	96	96	n/a	96	96
ECC1 heavyweight, integer	n/a	n/a	n/a	128	128	136
ECC1 heavyweight, floating-point	n/a	n/a	n/a	128	128	136
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	136
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	136
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	48	48	48	48	48	48
BCC1 lightweight, floating-point	56	56	56	56	56	56
BCC1 heavyweight, integer	96	96	96	96	96	96
BCC1 heavyweight, floating-point	96	96	96	96	96	96
BCC2 lightweight, integer	n/a	56	56	n/a	56	56
BCC2 lightweight, floating-point	n/a	56	56	n/a	56	56
BCC2 heavyweight, integer	n/a	104	96	n/a	104	96
BCC2 heavyweight, floating-point	n/a	104	96	n/a	104	96
ECC1 heavyweight, integer	n/a	n/a	n/a	136	136	136
ECC1 heavyweight, floating-point	n/a	n/a	n/a	136	136	136
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	136
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	136

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	56	56	64	56	56	64
BCC1 lightweight, floating-point	64	64	72	64	64	72
BCC1 heavyweight, integer	104	104	112	104	104	112
BCC1 heavyweight, floating-point	104	104	112	104	104	112
BCC2 lightweight, integer	n/a	64	72	n/a	64	72
BCC2 lightweight, floating-point	n/a	64	72	n/a	64	72
BCC2 heavyweight, integer	n/a	112	112	n/a	112	112
BCC2 heavyweight, floating-point	n/a	112	112	n/a	112	112
ECC1 heavyweight, integer	n/a	n/a	n/a	144	144	152
ECC1 heavyweight, floating-point	n/a	n/a	n/a	144	144	152
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	152
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	152
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	64	64	64	64	64	64
BCC1 lightweight, floating-point	72	72	72	72	72	72
BCC1 heavyweight, integer	112	112	112	112	112	112
BCC1 heavyweight, floating-point	112	112	112	112	112	112
BCC2 lightweight, integer	n/a	72	72	n/a	72	72
BCC2 lightweight, floating-point	n/a	72	72	n/a	72	72
BCC2 heavyweight, integer	n/a	120	112	n/a	120	112
BCC2 heavyweight, floating-point	n/a	120	112	n/a	120	112
ECC1 heavyweight, integer	n/a	n/a	n/a	152	152	152
ECC1 heavyweight, floating-point	n/a	n/a	n/a	152	152	152
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	152
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	152

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	56	56	64	56	56	64
BCC1 lightweight, floating-point	64	64	72	64	64	72
BCC1 heavyweight, integer	104	104	112	104	104	112
BCC1 heavyweight, floating-point	104	104	112	104	104	112
BCC2 lightweight, integer	n/a	64	72	n/a	64	72
BCC2 lightweight, floating-point	n/a	64	72	n/a	64	72
BCC2 heavyweight, integer	n/a	112	112	n/a	112	112
BCC2 heavyweight, floating-point	n/a	112	112	n/a	112	112
ECC1 heavyweight, integer	n/a	n/a	n/a	144	144	152
ECC1 heavyweight, floating-point	n/a	n/a	n/a	144	144	152
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	152
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	152
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	64	64	64	64	64	64
BCC1 lightweight, floating-point	72	72	72	72	72	72
BCC1 heavyweight, integer	112	112	112	112	112	112
BCC1 heavyweight, floating-point	112	112	112	112	112	112
BCC2 lightweight, integer	n/a	72	72	n/a	72	72
BCC2 lightweight, floating-point	n/a	72	72	n/a	72	72
BCC2 heavyweight, integer	n/a	120	112	n/a	120	112
BCC2 heavyweight, floating-point	n/a	120	112	n/a	120	112
ECC1 heavyweight, integer	n/a	n/a	n/a	152	152	152
ECC1 heavyweight, floating-point	n/a	n/a	n/a	152	152	152
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	152
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	152

Important: Sizes in API calls may be shorter. This is due to the sharing and caching of values in literal pools produced in the assembly language objects.

5 Extended Interrupt Control Functions

This section describes some extra functions provided by this port of RTA-OSEK to control interrupt handling. **Please note that these functions are not part of the OSEK standard.**

The functions discussed below may be called from tasks, Category 1 and 2 ISRs and OSEK hooks.

5.1 Globally Disabling and Enabling IRQ Interrupts

The following functions are provided to globally disable and enable IRQ interrupts.

```
void OSDisableIRQ(void);
```

This function sets bit 7 of the `CPSR` register (the I bit) to disable IRQ interrupts. The function does not take into account nesting.

```
void OSEnableIRQ(void);
```

This function clears bit 7 of the `CPSR` register (the I bit) to enable IRQ interrupts. The function does not take into account nesting.

```
void OSSuspendIRQ(void);
```

This function sets bit 7 of the `CPSR` register (the I bit) to disable IRQ interrupts. The function does take into account nesting. That is if `OSSuspendIRQ()` is called n times in succession then n calls to `OSResumeIRQ()` will be needed to re-enable IRQ interrupts.

```
void OSResumeIRQ(void);
```

This function clears bit 7 of the `CPSR` register (the I bit) to enable IRQ interrupts. The function does take into account nesting.

5.2 Globally Disabling and Enabling FIQ Interrupts

The following functions are provided to globally disable and enable FIQ interrupts.

```
void OSDisableFIQ(void);
```

This function sets bit 6 of the `CPSR` register (the F bit) to disable FIQ interrupts. The function does not take into account nesting.

```
void OSEnableFIQ(void);
```

This function clears bit 6 of the `CPSR` register (the F bit) to enable FIQ interrupts. The function does not take into account nesting.

```
void OSSuspendFIQ(void);
```

This function sets bit 6 of the `CPSR` register (the F bit) to disable FIQ interrupts. The function does take into account nesting. That is if `OSSuspendFIQ()` is called n times in succession then n calls to `OSResumeFIQ()` will be needed to re-enable FIQ interrupts.

```
void OSResumeFIQ(void);
```

This function clears bit 6 of the `CPSR` register (the F bit) to enable FIQ interrupts. The function does take into account nesting.

Important: see section 3.2.1 about a possible problem with globally disabling FIQ interrupts.

5.3 Disabling and Enabling VIM Channels

The following functions are provided to disable and enable individual VIM channels.

```
void OSDisableVIMChannel(osUIntType c);
```

This function disables VIM channel number `c` by clearing bit number `c` in the VIM's `REQMASK` register. The function does not take into account nesting.

```
void OSEnableVIMChannel(osUIntType c);
```

This function enables VIM channel number `c` by setting bit number `c` in the VIM's `REQMASK` register. The function does not take into account nesting.

```
void OSSuspendVIMChannel(
         osUIntType           c,
         osUIntType volatile * p);
```

This function disables VIM channel number `c` by clearing bit number `c` in the VIM's `REQMASK` register. The function does take into account nesting. `p` is a pointer to a channel specific nesting count variable provided by the application. The variable pointed to by `p` should be set to zero before `OSSupendVIMChannel()` is first called. If `OSSuspendVIMChannel(c, p)` is called n times in succession then n calls to `OSResumeVIMChannel(c, p)` (all with the same values of `c` and `p`) will be needed to re-enable the VIM channel.

```
void OSResumeVIMChannel(
         osUIntType           c,
         osUIntType volatile * p);
```

This function disables VIM channel number `c` by setting bit number `c` in the VIM's `REQMASK` register. The function does take into account nesting. See `OSSuspendVIMChannel()` for a discussion of the `p` argument.

Note: the application is required to provide the nesting count variables to save RAM. If the nesting count variable were provided by RTA-OSEK, RTA-OSEK would need to reserve a variable for each VIM channel when the `OSSuspendVIMChannel()` function were compiled. Since there are 64 VIM channels, few of which may actually be in use, a significant amount of RAM can be saved by having the application only define nesting count variables for the channels it wishes to control.

5.4 Rules for Using the Extended Interrupt Control Functions

Great care must be taken when using the extended interrupt control functions that change the values of the F and I bits in `CPSR` otherwise RTA-OSEK's assumptions about interrupt priority level may be subverted and RTA-OSEK may behave in undefined ways. The following table contains a list of the values that RTA-OSEK expects to be in the F (`CPSR` bit 6) and I (`CPSR` bit 7) bits when running tasks and ISRs. See also the comments in section 3.2.1.

Level	CPSR [7:6]	Description
0	00	User level – in task code – this is the version of user level normally used by RTA-OSEK and is valid inside API calls
0	01	User level – in task code – this version of user level is not valid inside API calls
1	10	Operating system level – in IRQ interrupt ISR code
2	11	Above operating system level – in FIQ interrupt ISR code

The following rules must be obeyed when using the extended interrupt control functions.

5.4.1 Tasks and ISRs Must Preserve F and I bit Settings

The F and I bits in `CPSR` must be the same at the end of an ISR or task as they were at the beginning.

For example the following code is legal:

```
TASK(my_task)
{
    OSDisableIRQ();

    /* More code. */

    OSEnableIRQ();

    TerminateTask();
}
```

The following code is not legal because the ISR terminates without having preserved the value of the I bit. When a Category 2 ISR starts running the I bit is set to disable further IRQ interrupts:

```
ISR(my_cat2_isr)
{
    OSDisableIRQ();

    /* More code. */

    OSEnableIRQ();
}
```

Important: most of the functionality offered by the extended interrupt control functions that change the F and I bits in CPSR are also provided in a much safer way by the standard OSEK API functions `DisableAllInterrupts()`, `EnableAllInterrupts()`, `SuspendAllInterrupts()`, `ResumeAllInterrupts()`, `SuspendOSInterrupts()` and `ResumeOSInterrupts()`.

5.4.2 Do Not Use RTA-OSEK APIs when the F and I bits Have Been Changed

If a task or ISR uses the extended interrupt control functions to change the value of the F or I bits in `CPSR` then the task or ISR must not call any standard RTA-OSEK API functions until it has restored the values of the F and I bits.

For example the following code is legal:

```
TASK(my_task)
{
    OSDisableIRQ();

    /* More code. */

    OSEnableIRQ();

    /* Call an OSEK API function e.g. */
    ActivateTask(my_other_task);

    TerminateTask();
}
```

However the following code is not legal:

```
TASK(my_task)
{
    OSDisableIRQ();

```

```
    /* Call an OSEK API function e.g. */
    ActivateTask(my_other_task);

    OSEnableIRQ();

    TerminateTask();
}
```

5.4.3 The `OSSuspendFIQ()` and `OSResumeFIQ()` are Not Re-Entrant

A higher priority task or ISR must not call `OSSuspendFIQ()` or `OSResumeFIQ()` while a lower priority task or ISR is already executing `OSSuspendFIQ()` or `OSResumeFIQ()`.

5.4.4 The `OSSuspendIRQ()` and `OSResumeIRQ()` are Not Re-Entrant

A higher priority task or ISR must not call `OSSuspendIRQ()` or `OSResumeIRQ()` while a lower priority task or ISR is already executing `OSSuspendIRQ()` or `OSResumeIRQ()`.

5.4.5 The `OSSuspendVIMChannel()` and `OSResumeVIMChannl()` are Not Re-Entrant

A higher priority task or ISR must not call `OSSuspendVIMChannel(c, p)` or `OSResumeVIMChannel(c, p)` while a lower priority task or ISR is already executing `OSSuspendVIMChannel(c, p)` or `OSResumeVIMChannel(c, p)` for the same values of `c` and `p`.

6 Implementing SWI Handlers

An RTA-OSEK application may implement SWI handlers. An SWI is a software interrupt that places the CPU in SVC mode and starts executing at vector 0x08 in the CPU vector table. More details can be found in the RealView Compilation Tools, Developer Guide section 6.6.

6.1 Handling an SWI

To handle an SWI in an RTA-OSEK application an SWI handler similar to the following is needed.

```
  AREA ||.text||, CODE, READONLY
  CODE32
  ALIGN 4

SWI_Handler PROC
  EXPORT SWI_Handler

  ; Store registers on the stack.
  STMFD  sp!,{r0-r4,r12,lr}

  ; Set r1 to point to the registers on
  ; the stack.
  MOV    r1,sp

  ; Get SPSR and store it on the stack.
  ; This is needed for nested SWIs.
  MRS    r0,SPSR
  STMFD  sp!,{r0}

  ; Work out the SWI number. We must allow
  ; for the caller being in ARM or THUMB
  ; mode.
  TST    r0,#0x20           ; THUMB caller?
  LDRNEH r0,[lr,#-2]        ; Yes: load half word
  BICNE  r0,r0,#0xFF00      ; Yes: get SWI number
  LDREQ  r0,[lr,#-4]        ; No: load word
  BICEQ  r0,r0,#0xFF000000 ; No: get SWI number

  ; At this point r0 contains the SWI number and
  ; r1 points to values of r0-r3 on the stack.

  ; If ATPCS is followed r4 is preserved by the
  ; called C function. The SVC stack must be made
  ; 8-byte aligned here for ATPCS compatibility.
  MOV    r4,sp
  BIC    sp,sp,#7

  ; Load the address of the C SWI handler.
```

```
  LDR    r2,Const_SWI_C_Handler

  ; Load the link register with the return
  ; addres.
  MOV    lr,pc

  ; Call the C handler function. This may be in
  ; ARM or THUMB.
  BX     r2

  ; Restore the stack pointer.
  MOV    sp,r4

  ; Restore SPSR.
  LDMFD  sp!,{r0}
  MSR    SPSR_cf,r0

  ; Restore other registers and return.
  LDMFD  sp!,{r0-r4,r12,pc}^

  ENDP

Const_SWI_C_Handler
  IMPORT SWI_C_Handler
  DCD    SWI_C_Handler

  END
```

A Category 1 ISR called `SWI_Handler` should be bound to the SWI vector (0x08). When an SWI occurs `SWI_Handler` will be run. `SWI_Handler` works out the SWI number and saves context on the stack and then calls the C function called `SWI_C_Handler`.

`SWI_C_Handler` should have a prototype like:

```
void SWI_C_Handler(unsigned   swi_number,
                   unsigned * regs);
```

When `SWI_C_Handler` is called `swi_number` will contain the SWI number used in the `SWI` instruction. The values of registers r0-r3 at the time the `SWI` instruction was executed will be stored on the stack pointed to by `regs`. `regs[n]` will access the value of register `rn`. If `SWI_C_Handler` changes the value of `regs[n]` then register `rn` will contain this new value when the CPU returns to the instruction after the `SWI` instruction.

6.2 Generating an SWI

SWIs can be generated from assembler by using the `SWI` instruction or from C by using the `__swi()` function modifier. See section 6.6.5 of the RealView Compilation Tools, Developer Guide. For example the following code fragment could be used:

```
__swi(0) void my_swi(unsigned, unsigned);

  /* Somewhere in application code. */
  my_swi(1, 2);
```

When `SWI_C_Handler` is called `swi_number` will contain the value 0, `regs[0]` will contain the value 1 and `regs[1]` will contain the value 2.

Support

For product support, please contact your local ETAS representative.

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