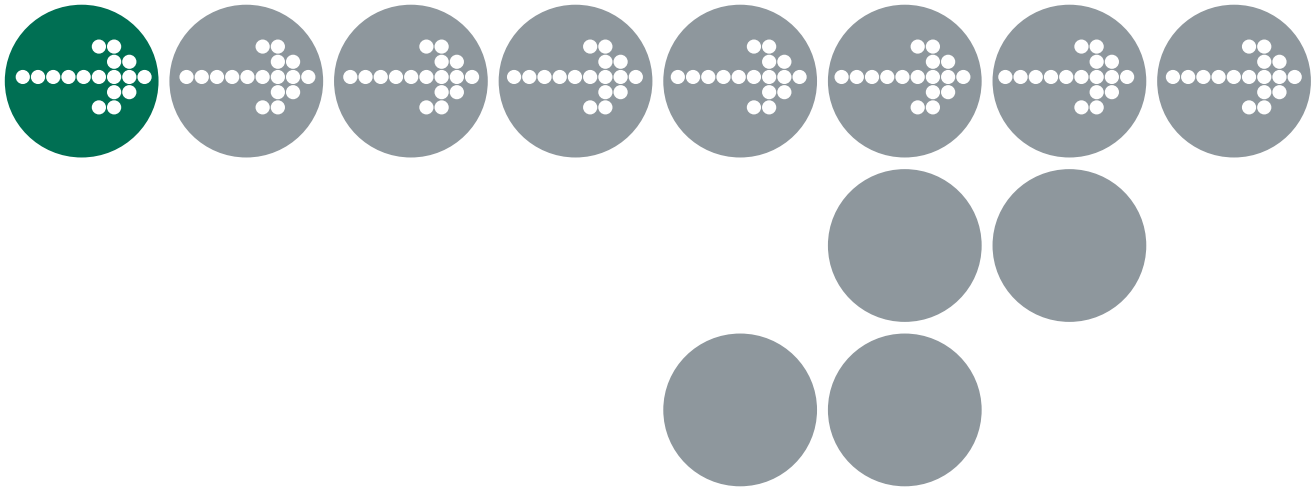
LiveDevices

Binding Manual: COSMIC ST7



realogy
Real-time Architect

Contact Details

Great Britain

LiveDevices Ltd.
Atlas House
Link Business Park
Osbaldwick Link Road
Osbaldwick
York
YO10 3JB

Tel.: +44 (0) 19 04 56 25 80
Fax: +44 (0) 19 04 56 25 81

www.livedevices.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

Japan

ETAS K.K.
9-1 Ushikubo 3-chome
Tsuzuki-ku
Yokohama 224-0012

Tel.: +81 (45) 912-95 50
Fax: +81 (45) 912-95 52

www.etas.co.jp

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

France

ETAS S.A.S.
1, place des Etats 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.
3F, Samseung Bldg. 61-1
Yangjae-dong, Seocho-gu
Seoul

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

Version: RM00039-001 .02

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

Real-Time Architect, RTA, RTArchitect, Realogy, the Time Compiler, SSX5 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 1-1
1.1 Who Should Read this Guide? 1-1
1.2 Conventions 1-1

2 Toolchain Issues 2-1
2.1 Memory Models 2-1
2.2 Compiler 2-1
2.3 Compiler Binary Versions 2-2
2.4 Access to 16-bit Registers 2-2
2.5 Assembler 2-2
2.6 Linker/Locator 2-3
2.7 Overriding the Default Stack Location 2-3
2.8 Debugger 2-4
2.8.1 Hitex HiTOP 2-4
2.8.2 Cosmic Zap 2-4

3 Target Hardware Issues 3-1
3.1 Interrupts 3-1
3.1.1 Interrupt Levels 3-1
3.1.2 Interrupt Vectors 3-1
3.1.3 Category 1 Handlers 3-1
3.1.4 Category 2 Handlers 3-2
3.1.5 Vector Table Issues 3-2
3.2 TRAP Instruction Priority 3-2
3.3 Correct Behavior when Updating CC 3-3
3.4 Support for ST7 Variants with Nested Interrupts 3-3
3.5 Register Settings 3-3
3.6 Stack Usage 3-3
3.6.1 Number of Stacks 3-3
3.6.2 Stack Usage within API Calls 3-4

4 Parameters of Implementation 4-1
4.1 Functionality 4-1
4.2 Hardware Resources 4-2
4.2.1 ROM and RAM Overheads 4-2
4.2.2 ROM and RAM for OSEK OS Objects 4-3
4.2.3 Size of Linkable Modules 4-7
4.2.4 Reserved Hardware Resources 4-19
4.3 Performance 4-19

4.3.1 Execution Times for SSX5 API Calls 4-19
4.3.2 OS Start-up Time 4-28
4.3.3 Interrupt Latencies 4-28
4.3.4 Task Switching Times 4-29
4.4 Configuration of Run-time Context 4-32

5 Use of @stack and @nostack 5-1

1 About this Guide

This guide provides port specific information for the ST7/COSMIC 16 task implementation of Realogy Real-Time Architect.

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 SSX5 on your target hardware. Port specific parameters of implementation are also provided, giving the RAM and ROM requirements for each SSX5 object and execution times for each SSX5 API call.

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 SSX5 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 SSX5.

In this guide you'll see that program code, header file names, C type names, C functions and SSX5 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 SSX5 and your toolchain. A port of SSX5 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 Memory Models

All SSX5 libraries have been compiled with a stack-based memory model and this memory model is enforced for tasks and callbacks.

Application routines may use any memory model (either stack-based or memory-based) with any pointer size. See Chapter 5 for further information on the use of memory-based memory models.

2.2 Compiler

SSX5 was built using the following compiler:

Vendor	Cosmic
Compiler	cxst7
Version	v4.4d (cgst7: v4.4f; cost7: v4.2p; clib: v4.2b)

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

Option	Description
`+debug`	For ORTI support to work correctly, the Cosmic ZAP debugger requires files to be compiled with debug information.

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

Option	Description
`-g -a`	Enable optimization of `_asm()` code. `_asm()` is used for the implementation of `OS_ATOMIC()` and automatic optimization may result in non-atomic operation.

The C file that RTA generates from your OIL configuration file is called `osekdefs.c`. This file defines configuration parameters for SSX5 when running your application.

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

Option	Description
`+debug`	For ORTI support to work correctly, the Cosmic ZAP debugger requires files to be compiled with debug information.

2.3 Compiler Binary Versions

The SSX5 libraries were compiled with v4.4f of `cgst7.exe` and v4.2p of `cost7.exe`. The libraries, however, are compatible with user applications compiled with v4.4d of `cgst7.exe` and v4.2m of `cost7.exe`.

2.4 Access to 16-bit Registers

Care is required when reading 16-bit timer counter values in C routines. In simple expressions, the compiler-generated code correctly reads the high byte of the timer counter first.

When the counter is subtracted from another value, however, the generated code references the low byte of the counter first. This can result in incorrect timer values being read. This is particularly relevant when writing callback functions for advanced schedules in SSX5.

For subtractions involving the timer value, an assembler function must be written to read the timer value high-byte first. The most significant byte must be read into the X register and the least significant byte must be read into the A register.

The following example shows an inline assembler function that could be used to read the ST72561 16-bit timer counter (at address 0x58):

```
(TimerType)_asm("ld x,0x58\n  ld a,0x59")
```

2.5 Assembler

SSX5 was built using the following assembler:

Vendor	Cosmic
Assembler	cast7
Version	v4.2r

The assembly file that RTA generates from your OIL configuration file is called `osgen.s`. This file defines configuration parameters for SSX5 when running your application.

2.6 Linker/Locator

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

Sections	ROM/RAM	Description
`os_pid`	ROM	SSX5 read-only data
`os_pird`	ROM	SSX5 initialization data
`os_pnird`	ROM	SSX5 zero-page initialization data
`os_pir`	RAM	SSX5 initialized data
`os_pnir`	Zero-page RAM	SSX5 zero-page initialized data
`os_pur`	RAM	SSX5 uninitialized data
`os_intvec`	ROM	Vector table if generated by RTArchitect

All Cosmic run-time libraries are compatible with SSX5.

Note, however, that any tasks that perform long (i.e. 32-bit) integer division must be declared as FP tasks in RTArchitect. This is because the Cosmic C library long integer division routines (`c_xldiv`, `c_ldiv`) use some temporary variables during calculation.

When an interrupt occurs or a task is set running, these variables are not preserved as part of the normal process context. Instead, they are only preserved as part of the SSX5 floating-point context. As a result of this, only those few tasks and Category 2 ISRs that perform long integer division incur the overhead of preserving these variables.

If your application uses 32-bit integer division and you supply alternative implementations of the routines in `osfptgt.c`, your routines must also preserve these temporary variables.

Note that Category 1 ISRs do not automatically preserve `c_lacc` or `c_lacc2`. These are the temporary variables used for floating-point and 32-bit integer arithmetic.

2.7 Overriding the Default Stack Location

The SSX5 libraries are built for variants with a 256-byte stack located at 0x100 to 0x1ff. It is possible to customize the libraries for a 128-byte stack by adding the following line at the start of the linker control file:

```
+def _os_stack_top=0x017f
```

This value tells SSX5 where the top of the stack is located. In other words, it specifies the reset value of the stack pointer. The example given above is, therefore, correct for a variant that has 128 bytes of stack located at 0x100 to 0x17f.

2.8 Debugger

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

ORTI Compatible Debuggers
Hitex HiTOP
Cosmic Zap v3.7

2.8.1 Hitex HiTOP

The HiTOP debugger requires a plug-in extension before it can support ORTI. This is available from Hitex.

2.8.2 Cosmic Zap

In order to support ORTI correctly, it is necessary to compile all application files with debug information.

To compile with debug from RTArchitect, include the following line in the Environment section of the Custom Build configuration:

```
APP_COPT=+debug
```

3 Target Hardware Issues

3.1 Interrupts

This section explains the implementation of the SSX5 interrupt model. You can find out more about configuring interrupts for SSX5 in the *RTA User Guide*.

3.1.1 Interrupt Levels

Interrupts, in SSX5, 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 User Guide*. The hardware interrupt controller is explained in the *ST7 Users Guide*.

The following table shows how SSX5 IPLs relate to interrupt priorities on the target hardware:

IPL	I Bits in Condition Code Register	Description
0	0x20	User level
1	0x28	OS level (Category 1 and 2 interrupts)

3.1.2 Interrupt Vectors

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

Vector	Legality
0xFFFE	RESET cannot be used

The valid base addresses for the vector table are:

Base Address	Notes
0xFFE0	

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 Cosmic C compiler can generate appropriate interrupt handling code for a C function decorated with the `@interrupt` function qualifier. You can find out more in your compiler documentation.

3.1.4 Category 2 Handlers

Category 2 ISRs are provided with a C function context by SSX5, since SSX5 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 SSX5.

3.1.5 Vector Table Issues

When you configure your application with RTArchitect 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 SSX5.

The following table shows the syntax for labels attached to SSX5 Category 2 interrupt handlers (`VVVV` represents the 4 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
`0xVVVV`	`_os_wrapper_VVVV`
eg : `0xFFF2`	`_os_wrapper_FFF2`

The SSX5 configuration system does not control the reset vector for the ST7. An appropriate vector must be created manually within the application source and must also be placed correctly during linking. You can find an example of this in the `reset.c` and `memory.lkf` files of the example application, which is installed as part of SSX5.

3.2 TRAP Instruction Priority

The ST7 `TRAP` instruction is a non-maskable interrupt source. The following limitations apply to the ISR for the `TRAP` interrupt:

- No OS APIs may be used.
- Even if the `TRAP` ISR is declared at low priority, it will preempt a high priority task or interrupt if a `TRAP` instruction is encountered.
- The `TRAP` interrupt cannot be disabled by any OS operation, including resource locks, `DisableAllInterrupts()`, `SuspendOSInterrupts()` and `SuspendAllInterrupts()` APIs.

3.3 Correct Behavior when Updating CC

The errata sheets for the ST72321J and ST72321R/AR variants indicate that, if an interrupt request occurs while a "`pop cc`" instruction is executed, some revisions will fail to correctly detect the source of interrupt.

The suggested workaround is to put a "`sim`" instruction before the "`pop cc`". The SSX5 libraries implement this workaround and are, therefore, compatible with those variants that exhibit the problem.

3.4 Support for ST7 Variants with Nested Interrupts

Some variants of the ST7 have two interrupt mask bits in the Condition Code Register (thereby supporting nested interrupts), while others have only a single interrupt mask bit. This version of SSX5 supports a single level of interrupt and is compatible with both versions of the interrupt controller.

When running on an ST7 variant that offers nested interrupts, it is your responsibility to configure the interrupt priority registers to ensure that all interrupts are configured to occur at the maximum priority (i.e. with I1=1, I0=1).

To do this on an ST72561 or ST72521, for example, 0xFF must be written to `ISPR0`, `ISPR1`, `ISPR2` and `ISPR3`.

3.5 Register Settings

SSX5 does not require the initialization of registers before calling `StartOS()`.

SSX5 uses the following hardware registers. They should not be altered by user code.

Register (Field)	Notes
CCR (`I0` and `I1`, or `I`)	The interrupt enable bits (`I0` and `I1` in processors that support nested interrupts and `I` in single level variants) must be manipulated entirely by OS APIs.

3.6 Stack Usage

3.6.1 Number of Stacks

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

`StackOffsetType` is a scalar, representing the number of bytes on the stack, with C type: `unsigned char`

3.6.2 Stack Usage within API Calls

The maximum stack usage within SSX5 API calls, excluding calls to hooks and callbacks, is as follows:

Standard

API max usage (bytes): 16

Timing

API max usage (bytes): 16

Extended

API max usage (bytes): 22

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.

4 Parameters of Implementation

This chapter provides detailed information on the functionality, performance and memory demands of SSX5.

SSX5 is highly scalable. As a result, different figures will be obtained when your application uses different sets of features. These feature-sets give 6 classes of SSX5, 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	**ST72561**
Clock speed (MHz)	4
Code memory	AX6811 ICE emulated memory
Read-only data memory	AX6811 ICE emulated memory
Read-write data memory	AX6811 ICE emulated memory

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	16	16	16	16	16	16
Maximum number of not suspended tasks	16	16	16	16	16	16
Maximum number of priorities	16	16	16	16	16	16
Number of tasks per priority (for BCC2 and ECC2)	n/a	16	16	n/a	16	16
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	8	8	8
Limits for the number of alarm objects (per system / per task)	not limited by SSX5					
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 SSX5					
Limits for the number of nested resources (per system / per task)	255	255	255	255	255	255
Limits for the number of application modes (per system)	255					

4.2 Hardware Resources

4.2.1 ROM and RAM Overheads

The following tables give the ROM and RAM overheads for SSX5 (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 / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	12	12	12	12	12	12
	ROM	118	118	118	118	118	118
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	22	22	22	22	22	22
	ROM	153	153	153	153	153	153
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	28	28	28	28	28	28
	ROM	173	173	173	173	173	173
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

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. SSX5 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 SSX5. (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	17	17	17	17	17	17
BCC1 Heavyweight task	RAM	3	3	3	3	3	3
	ROM	19	19	19	19	19	19
BCC2 task	RAM	n/a	4	6	n/a	4	6
	ROM	n/a	20	24	n/a	20	24
ECC1, Integer task	RAM	n/a	n/a	n/a	8	8	8
	ROM	n/a	n/a	n/a	27	27	27
ECC1, floating-point task	RAM	n/a	n/a	n/a	16	16	16
	ROM	n/a	n/a	n/a	27	27	27
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	10
	ROM	n/a	n/a	n/a	n/a	n/a	31
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	18
	ROM	n/a	n/a	n/a	n/a	n/a	31
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	28	28	28	28	28	28
Category 2 ISR, floating-point	RAM	8	8	8	8	8	8
	ROM	37	37	37	37	37	37

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
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	10	10	10	10	10	10
Alarm	RAM	5	5	5	5	5	5
	ROM	36	36	36	36	36	36
Counter	RAM	2	2	2	2	2	2
	ROM	37	37	37	37	37	37
Message	RAM	11	11	11	31	31	31
	ROM	10	10	10	26	26	26
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	10	10	10	10	10	10
Event	RAM	0	0	0	0	0	0
	ROM	1	1	1	1	1	1
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	7	7	7	7	7	7
	ROM	16	16	16	16	16	16
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

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	5	5	5	5	5	5
	ROM	22	22	22	22	22	22
BCC1 Heavyweight task	RAM	8	8	8	8	8	8
	ROM	24	24	24	24	24	24
BCC2 task	RAM	n/a	9	11	n/a	9	11
	ROM	n/a	25	29	n/a	25	29
ECC1, Integer task	RAM	n/a	n/a	n/a	13	13	13
	ROM	n/a	n/a	n/a	32	32	32
ECC1, floating-point task	RAM	n/a	n/a	n/a	21	21	21
	ROM	n/a	n/a	n/a	32	32	32
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	15
	ROM	n/a	n/a	n/a	n/a	n/a	36
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	23
	ROM	n/a	n/a	n/a	n/a	n/a	36
Category 2 ISR	RAM	5	5	5	5	5	5
	ROM	46	46	46	46	46	46
Category 2 ISR, floating-point	RAM	13	13	13	13	13	13
	ROM	52	52	52	52	52	52
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
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	10	10	10	10	10	10
Alarm	RAM	5	5	5	5	5	5
	ROM	36	36	36	36	36	36
Counter	RAM	2	2	2	2	2	2
	ROM	37	37	37	37	37	37
Message	RAM	11	11	11	31	31	31
	ROM	10	10	10	26	26	26
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	10	10	10	10	10	10
Event	RAM	0	0	0	0	0	0
	ROM	1	1	1	1	1	1
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	7	7	7	7	7	7
	ROM	16	16	16	16	16	16
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

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	6	6	6	6	6	6
	ROM	26	26	26	26	26	26
BCC1 Heavyweight task	RAM	9	9	9	9	9	9
	ROM	26	26	26	26	26	26
BCC2 task	RAM	n/a	10	12	n/a	10	12
	ROM	n/a	27	31	n/a	27	31
ECC1, Integer task	RAM	n/a	n/a	n/a	14	14	14
	ROM	n/a	n/a	n/a	34	34	34
ECC1, floating-point task	RAM	n/a	n/a	n/a	22	22	22
	ROM	n/a	n/a	n/a	34	34	34
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	16
	ROM	n/a	n/a	n/a	n/a	n/a	38
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	24
	ROM	n/a	n/a	n/a	n/a	n/a	38
Category 2 ISR	RAM	6	6	6	6	6	6
	ROM	50	50	50	50	50	50
Category 2 ISR, floating-point	RAM	14	14	14	14	14	14
	ROM	56	56	56	56	56	56
Resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Linked resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Alarm	RAM	5	5	5	5	5	5
	ROM	38	38	38	38	38	38
Counter	RAM	2	2	2	2	2	2
	ROM	39	39	39	39	39	39
Message	RAM	11	11	11	31	31	31
	ROM	12	12	12	28	28	28
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	3	3	3	3	3	3
	ROM	14	14	14	14	14	14
Event	RAM	0	0	0	0	0	0
	ROM	1	1	1	1	1	1
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	5	0	5	5
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (writable)	RAM	10	10	10	10	10	10
	ROM	10	10	10	10	10	10
Schedule	RAM	9	9	9	9	9	9
	ROM	20	20	20	20	20	20
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

4.2.3 Size of Linkable Modules

SSX5 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 SSX5 OS status types (standard, timing, and extended).

In some cases there are multiple variants of particular API calls. This is because the offline configuration of SSX5 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 SSX5 for which the call is valid.

The call variants are as follows:

Variant	Description
1i	Idle task is only ECC task.
CCCA	OSEK COM class.
CCCB	OSEK COM class.
CLEx	Resource tests in Extended OS Status.
fp	ECC task uses floating-point.
H	Used for heavyweight termination only.
Hook	Pre- and Post- Task hooks are used.
KL	API is called from OS level.
KL1i	API is called from OS level, idle task is only ECC task.
KL2	Activated taskset has one BCC2 task.
LExt	Used for lightweight termination in Extended Status.
ServiceID	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
Parameters	`ErrorHook` uses `GetServiceID` and `GetServiceParameters`.
NoHook	Pre- and/or Post- Task hooks are not used.
NS	No context switch is possible.
NS1i	No context switch is possible, idle task is only ECC task.
NS2	Activated taskset has one BCC2 task.
NSH	Chain from heavyweight task, not to higher priority.
NSL	Chain from lightweight task, not to higher priority.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.
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	121	289	508	153	321	603
	NS		105	273	492	137	305	587
	KL	2	86	254	473	118	286	568
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	25	25	25	25	25	25
ChainTask	SWL	1, 8	122	291	503	154	323	598
	SWH	1, 9	158	328	544	193	363	639
	NSL	8	122	291	503	154	323	598
	NSH	9	150	319	536	185	354	631

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Schedule			75	75	149	75	75	149
GetTaskID			55	55	55	55	55	55
GetTaskState			97	97	97	144	144	144
EnableAllInterrupts			6	6	6	6	6	6
DisableAllInterrupts			1	1	1	1	1	1
ResumeAllInterrupts			15	15	15	15	15	15
SuspendAllInterrupts			1	1	1	1	1	1
ResumeOSInterrupts			16	16	16	16	16	16
SuspendOSInterrupts			1	1	1	1	1	1
GetResource	Task	7	77	77	90	77	77	90
	Combined	6	149	149	149	149	149	149
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	55	55	55	55	55	55
	Combined	6	132	132	132	132	132	132
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	192	192	544
	NS		n/a	n/a	n/a	176	176	528
	NS1i	10	n/a	n/a	n/a	48	n/a	n/a
	KL	2	n/a	n/a	n/a	161	161	513
	KL1i	2, 10	n/a	n/a	n/a	39	n/a	n/a
ClearEvent			n/a	n/a	n/a	42	42	42
GetEvent			n/a	n/a	n/a	42	42	42
WaitEvent	<default>		n/a	n/a	n/a	407	407	793
	fp	11	n/a	n/a	n/a	493	493	966
	1i	10	n/a	n/a	n/a	35	n/a	n/a
GetAlarmBase			58	58	58	58	58	58
GetAlarm			190	190	190	190	190	190
SetRelAlarm			226	226	226	226	226	226
SetAbsAlarm			347	347	347	347	347	347
CancelAlarm			98	98	98	98	98	98
InitCounter			123	123	123	123	123	123
GetCounterValue			189	189	189	189	189	189
osek_tick_alarm	<default>		195	195	195	195	195	195
	KL	2	169	169	169	169	169	169
osek_incr_counter			207	207	207	207	207	207
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			120	120	120	120	120	120
ShutdownOS	NoHook	12	5	5	5	5	5	5
	Hook	13	13	13	13	13	13	13
InitCOM			2	2	2	2	2	2

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
CloseCOM			2	2	2	2	2	2
StartCOM			19	19	19	19	19	19
StopCOM			21	21	21	21	21	21
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	112	112	112	636	636	636
	CCCB	15	636	636	636	636	636	636
GetMessageResource			67	67	67	67	67	67
ReleaseMessageResource			57	57	57	57	57	57
GetMessageStatus			140	140	140	140	140	140
SendMessage	SW CCCA	1, 14	194	194	194	739	739	739
	SW CCCB	1, 15	676	676	676	739	739	739
	NS CCCA	14	194	194	194	739	739	739
	NS CCCB	15	676	676	676	739	739	739
	KL CCCA	2, 14	177	177	177	723	723	723
	KL CCCB	2, 15	660	660	660	723	723	723
main_dispatch	NoHook	12	184	184	307	184	184	307
	Hook	13	227	227	350	227	227	350
sub_dispatch	B1LF	19	30	30	30	30	30	30
	B1HI	20	81	81	81	81	81	81
	B1HF	21	87	87	87	87	87	87
	B2LI	22	n/a	86	194	n/a	86	194
	B2LF	23	n/a	91	200	n/a	91	200
	B2HI	24	n/a	142	464	n/a	142	464
	B2HF	25	n/a	147	470	n/a	147	470
	E1HI	26	n/a	n/a	n/a	502	502	816
	E1HF	27	n/a	n/a	n/a	508	508	822
	E2HI	28	n/a	n/a	n/a	n/a	n/a	816
	E2HF	29	n/a	n/a	n/a	n/a	n/a	822
ErrorHook support		16	39	39	39	39	39	39
	ServiceID	17	45	45	45	45	45	45
	Parameters	18	81	81	81	81	81	81
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	52	195	444	85	249	513
	NS		36	179	428	69	233	497
	KL	2	17	164	417	50	222	486
ChainTaskset	SWL	1, 8	73	219	468	73	244	508
	SWH	1, 9	136	311	560	136	336	600
	NSL	8	73	219	468	73	244	508
	NSH	9	128	302	551	128	327	591

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetTasksetRef			44	44	44	44	44	44
MergeTaskset			84	84	84	84	84	84
AssignTaskset			44	44	44	44	44	44
RemoveTaskset			86	86	86	86	86	86
TestSubTaskset			121	121	121	121	121	121
TestEquivalentTaskset			82	82	82	82	82	82
TickSchedule	SW	1	401	389	389	389	389	389
	NS		385	373	373	373	373	373
	KL	2	369	357	357	357	357	357
AdvanceSchedule	SW	1	271	259	259	259	259	259
	NS		250	238	238	238	238	238
	KL	2	230	218	218	218	218	218
StartSchedule			172	172	172	172	172	172
StopSchedule			85	85	85	85	85	85
GetScheduleStatus			236	236	236	236	236	236
GetScheduleValue			105	105	105	105	105	105
GetScheduleNext			50	50	50	50	50	50
SetScheduleNext			48	48	48	48	48	48
GetArrivalpointDelay			50	50	50	50	50	50
SetArrivalpointDelay			48	48	48	48	48	48
GetArrivalpointTasksetRef			39	39	39	39	39	39
GetArrivalpointNext			50	50	50	50	50	50
SetArrivalpointNext			48	48	48	48	48	48
TestArrivalpointWritable			47	47	47	47	47	47
GetExecutionTime			3	3	3	3	3	3
GetLargestExecutionTime			25	25	25	25	25	25
ResetLargestExecutionTime			2	2	2	2	2	2
GetStackOffset			32	32	32	32	32	32
Floating-point support			350	350	350	350	350	350
Interrupt support			43	43	43	43	43	43
Interrupt support			100	100	100	100	100	100
Heavyweight/Extended Utility functions			27	27	27	27	27	27
Heavyweight/Extended Utility functions			29	29	29	29	29	29
Utility functions			68	68	68	68	68	68
Utility functions			106	106	106	106	106	106

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	121	289	508	153	321	603
	NS		105	273	492	137	305	587
	KL	2	86	254	473	118	286	568
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	25	25	25	25	25	25
ChainTask	SWL	1, 8	122	291	503	154	323	598
	SWH	1, 9	158	328	544	193	363	639
	NSL	8	122	291	503	154	323	598
	NSH	9	150	319	536	185	354	631
Schedule			93	93	167	93	93	167
GetTaskID			55	55	55	55	55	55
GetTaskState			97	97	97	144	144	144
EnableAllInterrupts			6	6	6	6	6	6
DisableAllInterrupts			1	1	1	1	1	1
ResumeAllInterrupts			15	15	15	15	15	15
SuspendAllInterrupts			1	1	1	1	1	1
ResumeOSInterrupts			16	16	16	16	16	16
SuspendOSInterrupts			1	1	1	1	1	1
GetResource	Task	7	77	77	90	77	77	90
	Combined	6	149	149	149	149	149	149
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	73	73	73	73	73	73
	Combined	6	168	168	168	168	168	168
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	192	192	544
	NS		n/a	n/a	n/a	176	176	528
	NS1i	10	n/a	n/a	n/a	48	n/a	n/a
	KL	2	n/a	n/a	n/a	161	161	513
	KL1i	2, 10	n/a	n/a	n/a	39	n/a	n/a
ClearEvent			n/a	n/a	n/a	42	42	42
GetEvent			n/a	n/a	n/a	42	42	42
WaitEvent	<default>		n/a	n/a	n/a	407	407	793
	fp	11	n/a	n/a	n/a	493	493	966
	1i	10	n/a	n/a	n/a	35	n/a	n/a
GetAlarmBase			58	58	58	58	58	58
GetAlarm			190	190	190	190	190	190
SetRelAlarm			226	226	226	226	226	226

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
SetAbsAlarm			347	347	347	347	347	347
CancelAlarm			98	98	98	98	98	98
InitCounter			123	123	123	123	123	123
GetCounterValue			189	189	189	189	189	189
osek_tick_alarm	<default>		195	195	195	195	195	195
	KL	2	169	169	169	169	169	169
osek_incr_counter			207	207	207	207	207	207
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			184	184	184	184	184	184
ShutdownOS	NoHook	12	5	5	5	5	5	5
	Hook	13	13	13	13	13	13	13
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			19	19	19	19	19	19
StopCOM			21	21	21	21	21	21
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	112	112	112	636	636	636
	CCCB	15	636	636	636	636	636	636
GetMessageResource			67	67	67	67	67	67
ReleaseMessageResource			57	57	57	57	57	57
GetMessageStatus			140	140	140	140	140	140
SendMessage	SW CCCA	1, 14	194	194	194	739	739	739
	SW CCCB	1, 15	676	676	676	739	739	739
	NS CCCA	14	194	194	194	739	739	739
	NS CCCB	15	676	676	676	739	739	739
	KL CCCA	2, 14	177	177	177	723	723	723
	KL CCCB	2, 15	660	660	660	723	723	723
main_dispatch	NoHook	12	401	401	521	401	401	521
	Hook	13	439	439	559	439	439	559
sub_dispatch	B1LF	19	9	9	9	9	9	9
	B1HI	20	70	70	70	70	70	70
	B1HF	21	77	77	77	77	77	77
	B2LI	22	n/a	64	172	n/a	64	172
	B2LF	23	n/a	69	178	n/a	69	178
	B2HI	24	n/a	130	452	n/a	130	452
	B2HF	25	n/a	135	458	n/a	135	458
	E1HI	26	n/a	n/a	n/a	566	566	880
	E1HF	27	n/a	n/a	n/a	572	572	886
	E2HI	28	n/a	n/a	n/a	n/a	n/a	880
	E2HF	29	n/a	n/a	n/a	n/a	n/a	886

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
ErrorHook support		16	39	39	39	39	39	39
	ServiceID	17	45	45	45	45	45	45
	Parameters	18	81	81	81	81	81	81
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	86	86	86	86	86	86
Timing_termination		4	183	183	183	183	183	183
ActivateTaskset	SW	1	52	195	444	85	249	513
	NS		36	179	428	69	233	497
	KL	2	17	164	417	50	222	486
ChainTaskset	SWL	1, 8	73	219	468	73	244	508
	SWH	1, 9	136	311	560	136	336	600
	NSL	8	73	219	468	73	244	508
	NSH	9	128	302	551	128	327	591
GetTasksetRef			44	44	44	44	44	44
MergeTaskset			84	84	84	84	84	84
AssignTaskset			44	44	44	44	44	44
RemoveTaskset			86	86	86	86	86	86
TestSubTaskset			121	121	121	121	121	121
TestEquivalentTaskset			82	82	82	82	82	82
TickSchedule	SW	1	401	389	389	389	389	389
	NS		385	373	373	373	373	373
	KL	2	369	357	357	357	357	357
AdvanceSchedule	SW	1	271	259	259	259	259	259
	NS		250	238	238	238	238	238
	KL	2	230	218	218	218	218	218
StartSchedule			172	172	172	172	172	172
StopSchedule			85	85	85	85	85	85
GetScheduleStatus			236	236	236	236	236	236
GetScheduleValue			105	105	105	105	105	105
GetScheduleNext			50	50	50	50	50	50
SetScheduleNext			48	48	48	48	48	48
GetArrivalpointDelay			50	50	50	50	50	50
SetArrivalpointDelay			48	48	48	48	48	48
GetArrivalpointTasksetRef			39	39	39	39	39	39
GetArrivalpointNext			50	50	50	50	50	50
SetArrivalpointNext			48	48	48	48	48	48
TestArrivalpointWritable			47	47	47	47	47	47
GetExecutionTime			146	146	146	146	146	146
GetLargestExecutionTime			61	61	61	61	61	61
ResetLargestExecutionTime			53	53	53	53	53	53

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetStackOffset			32	32	32	32	32	32
Floating-point support			350	350	350	350	350	350
Interrupt support			306	306	306	306	306	306
Interrupt support			100	100	100	100	100	100
Heavyweight/Extended Utility functions			27	27	27	27	27	27
Heavyweight/Extended Utility functions			29	29	29	29	29	29
Utility functions			68	68	68	68	68	68
Utility functions			106	106	106	106	106	106

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	239	410	624	275	444	720
	NS		430	600	809	460	628	904
	KL	2	202	374	590	239	409	686
TerminateTask	LExt	3	116	116	116	116	116	116
	H	5	159	159	159	159	159	159
ChainTask	SWL	1, 8	329	502	710	362	532	805
	SWH	1, 9	373	548	757	407	577	852
	NSL	8	537	708	914	567	736	1009
	NSH	9	568	740	948	599	768	1043
Schedule			190	190	265	190	190	265
GetTaskID			72	72	72	72	72	72
GetTaskState			263	263	263	286	286	286
EnableAllInterrupts			16	16	16	16	16	16
DisableAllInterrupts			1	1	1	1	1	1
ResumeAllInterrupts			48	48	48	48	48	48
SuspendAllInterrupts			1	1	1	1	1	1
ResumeOSInterrupts			49	49	49	49	49	49
SuspendOSInterrupts			1	1	1	1	1	1
GetResource	Task	7	712	712	588	712	712	588
	Combined	6	569	569	569	569	569	569
	CLEx	3	565	565	565	565	565	565
ReleaseResource	Task	7	537	537	537	537	537	537
	Combined	6	579	579	579	579	579	579
	CLEx	3	488	488	488	488	488	488

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
SetEvent	SW	1	n/a	n/a	n/a	375	375	725
	NS		n/a	n/a	n/a	553	553	904
	NS1i	10	n/a	n/a	n/a	222	n/a	n/a
	KL	2	n/a	n/a	n/a	335	335	687
	KL1i	2, 10	n/a	n/a	n/a	202	n/a	n/a
ClearEvent			n/a	n/a	n/a	122	122	122
GetEvent			n/a	n/a	n/a	258	258	258
WaitEvent	<default>		n/a	n/a	n/a	550	550	916
	fp	11	n/a	n/a	n/a	636	636	1089
	1i	10	n/a	n/a	n/a	168	n/a	n/a
GetAlarmBase			143	143	143	143	143	143
GetAlarm			230	230	230	230	230	230
SetRelAlarm			408	408	408	408	408	408
SetAbsAlarm			544	544	544	544	544	544
CancelAlarm			141	141	141	141	141	141
InitCounter			242	242	242	242	242	242
GetCounterValue			277	277	277	277	277	277
osek_tick_alarm	<default>		200	200	200	200	200	200
	KL	2	169	169	169	169	169	169
osek_incr_counter			207	207	207	207	207	207
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			194	194	194	194	194	194
ShutdownOS	NoHook	12	10	10	10	10	10	10
	Hook	13	18	18	18	18	18	18
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			29	29	29	29	29	29
StopCOM			36	36	36	36	36	36
ReadFlag			29	29	29	29	29	29
ResetFlag			22	22	22	22	22	22
ReceiveMessage	CCCA	14	169	169	169	694	694	694
	CCCB	15	694	694	694	694	694	694
GetMessageResource			114	114	114	114	114	114
ReleaseMessageResource			114	114	114	114	114	114
GetMessageStatus			175	175	175	175	175	175
SendMessage	SW CCCA	1, 14	253	253	253	798	798	798
	SW CCCB	1, 15	735	735	735	798	798	798
	NS CCCA	14	253	253	253	798	798	798
	NS CCCB	15	735	735	735	798	798	798
	KL CCCA	2, 14	247	247	247	793	793	793
	KL CCCB	2, 15	730	730	730	793	793	793

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
main_dispatch	NoHook	12	401	401	521	401	401	521
	Hook	13	439	439	559	439	439	559
sub_dispatch	B1LF	19	9	9	9	9	9	9
	B1HI	20	74	74	74	74	74	74
	B1HF	21	81	81	81	81	81	81
	B2LI	22	n/a	64	172	n/a	64	172
	B2LF	23	n/a	69	178	n/a	69	178
	B2HI	24	n/a	134	456	n/a	134	456
	B2HF	25	n/a	139	462	n/a	139	462
	E1HI	26	n/a	n/a	n/a	570	570	884
	E1HF	27	n/a	n/a	n/a	576	576	890
	E2HI	28	n/a	n/a	n/a	n/a	n/a	884
	E2HF	29	n/a	n/a	n/a	n/a	n/a	890
ErrorHook support		16	49	49	49	49	49	49
	ServiceID	17	55	55	55	55	55	55
	Parameters	18	91	91	91	91	91	91
validity_checks		3	104	104	104	104	104	104
Timing_dispatch		4	86	86	86	86	86	86
Timing_termination		4	183	183	183	183	183	183
ActivateTaskset	SW	1	439	609	852	498	691	963
	NS		600	770	1013	659	852	1124
	KL	2	405	575	818	464	657	929
ChainTaskset	SWL	1, 8	620	790	1033	649	843	1115
	SWH	1, 9	690	898	1141	719	951	1223
	NSL	8	797	967	1210	826	1020	1292
	NSH	9	859	1066	1309	888	1119	1391
GetTasksetRef			191	191	191	191	191	191
MergeTaskset			283	283	283	283	283	283
AssignTaskset			234	234	234	234	234	234
RemoveTaskset			285	285	285	285	285	285
TestSubTaskset			322	322	322	322	322	322
TestEquivalentTaskset			281	281	281	281	281	281
TickSchedule	SW	1	718	555	555	555	555	555
	NS		932	793	793	793	793	793
	KL	2	684	521	521	521	521	521
AdvanceSchedule	SW	1	643	465	465	465	465	465
	NS		856	717	717	717	717	717
	KL	2	609	430	430	430	430	430
StartSchedule			345	345	345	345	345	345
StopSchedule			167	167	167	167	167	167

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetScheduleStatus			333	333	333	333	333	333
GetScheduleValue			183	183	183	183	183	183
GetScheduleNext			135	135	135	135	135	135
SetScheduleNext			266	266	266	266	266	266
GetArrivalpointDelay			175	175	175	175	175	175
SetArrivalpointDelay			218	218	218	218	218	218
GetArrivalpointTasksetRef			164	164	164	164	164	164
GetArrivalpointNext			175	175	175	175	175	175
SetArrivalpointNext			306	306	306	306	306	306
TestArrivalpointWritable			170	170	170	170	170	170
GetExecutionTime			160	160	160	160	160	160
GetLargestExecutionTime			146	146	146	146	146	146
ResetLargestExecutionTime			124	124	124	124	124	124
GetStackOffset			32	32	32	32	32	32
Floating-point support			350	350	350	350	350	350
Interrupt support			306	306	306	306	306	306
Interrupt support			100	100	100	100	100	100
Heavyweight/Extended Utility functions			27	27	27	27	27	27
Heavyweight/Extended Utility functions			29	29	29	29	29	29
Utility functions			68	68	68	68	68	68
Utility functions			106	106	106	106	106	106

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.

Number	Note
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.
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 SSX5.

4.3 Performance

The collection of performance data for the ST7/COSMIC 16 task port of SSX5 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 SSX5 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	296	626	1182	358	598	1332
	NS	276	608	1162	340	578	1308
	KL	244	574	1128	306	544	1274
TerminateTask	LExt	0	0	0	0	0	0
	H	350	350	364	360	358	364
ChainTask	SWL	826	1158	2072	1150	1390	2476
	SWH	980	1312	2218	1296	1536	2626
	NSL	826	1156	2070	1152	1390	2476
	NSH	968	1300	2204	1284	1524	2612
Schedule	SW	248	248	358	248	248	360
GetTaskID		154	154	154	152	152	152
GetTaskState		256	256	256	368	368	370
EnableAllInterrupts		26	26	26	26	26	26
DisableAllInterrupts		30	28	30	30	28	28
ResumeAllInterrupts		40	44	40	40	40	42
SuspendAllInterrupts		48	48	48	48	50	48
ResumeOSInterrupts		42	42	42	42	42	40
SuspendOSInterrupts		48	48	48	48	48	48
GetResource	Task	276	278	302	276	276	302
	Combined	202	200	202	204	202	200
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	176	174	176	176	176	174
	Combined	220	220	220	220	220	220
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	426	424	424
	NS	n/a	n/a	n/a	424	426	424
	KL	n/a	n/a	n/a	392	392	392
ClearEvent		n/a	n/a	n/a	120	120	120
GetEvent		n/a	n/a	n/a	152	154	152
WaitEvent	<default>	n/a	n/a	n/a	1326	1326	1708
	fp	n/a	n/a	n/a	1388	1388	1774
GetAlarmBase		336	336	338	338	336	336
GetAlarm		480	482	482	480	480	480
SetRelAlarm		560	560	560	560	560	560
SetAbsAlarm		524	524	524	524	524	524
CancelAlarm		222	222	224	224	222	222
InitCounter		268	266	266	268	268	268
GetCounterValue		302	302	302	302	302	302
osek_tick_alarm	<default>	530	530	530	530	530	530
	KL	480	480	478	478	480	480

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
osek_incr_counter		160	162	162	160	160	160
GetActiveApplicationMode		12	12	12	12	12	12
StartOS		2660	2660	2660	2660	2660	2660
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	32	34	36	34	34	32
InitCOM		18	18	18	18	18	18
CloseCOM		18	18	18	18	18	22
StartCOM		68	68	66	822	822	824
StopCOM		52	50	50	52	50	52
ReadFlag		n/a	n/a	n/a	16	16	18
ResetFlag		n/a	n/a	n/a	16	16	16
ReceiveMessage		180	180	180	1588	1588	1624
GetMessageResource		n/a	n/a	n/a	558	558	564
ReleaseMessageResource		n/a	n/a	n/a	414	414	420
GetMessageStatus		n/a	n/a	n/a	350	350	362
SendMessage	SW	632	962	1516	1878	2116	2884
	NS	612	942	1496	1858	2098	2860
	KL	542	872	1426	1790	2028	2792
ActivateTaskset	SW	142	1818	2678	210	1792	2760
	NS	122	1798	2662	192	1772	2744
	KL	76	1766	2632	144	1746	2714
	SW2	142	1818	2678	210	1792	2762
	NS2	122	1798	2658	190	1772	2740
	KL2	74	1764	2630	142	1746	2718
ChainTaskset	SWL	804	2478	3712	1064	2668	3998
	SWH	1000	2664	3894	1262	2852	4182
	NSL	812	2488	3718	1074	2676	4006
	NSH	992	2652	3882	1254	2840	4174
GetTasksetRef		188	188	188	188	188	188
MergeTaskset		300	298	298	298	298	298
AssignTaskset		184	184	184	184	184	184
RemoveTaskset		304	306	306	306	306	306
TestSubTaskset		374	372	372	372	372	372
TestEquivalentTaskset		270	270	270	270	270	270
TickSchedule	SW	1294	3000	3866	1378	3096	4072
	NS	1274	2980	3848	1356	3074	4052
	KL	1240	2944	3810	1322	3040	4016
	SW2	1294	3000	3864	1376	2980	3952
	NS2	1274	2978	3846	1358	2960	3928
	KL2	1238	2944	3810	1322	2926	3892

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
AdvanceSchedule	SW	756	2460	3330	838	2558	3536
	NS	728	2434	3300	812	2530	3508
	KL	684	2388	3254	766	2484	3460
	SW2	756	2460	3326	840	2442	3410
	NS2	728	2434	3300	812	2414	3384
	KL2	682	2388	3258	766	2370	3340
StartSchedule		388	388	388	388	388	388
StopSchedule		248	248	248	248	248	248
GetScheduleStatus		526	526	526	526	526	526
GetScheduleValue		264	264	264	264	264	264
GetScheduleNext		220	220	220	220	222	222
SetScheduleNext		200	200	200	200	198	198
GetArrivalpointDelay		192	192	192	192	192	192
SetArrivalpointDelay		170	170	170	170	168	168
GetArrivalpointTasksetRef		148	150	148	148	150	150
GetArrivalpointNext		192	190	192	192	190	190
SetArrivalpointNext		168	172	168	168	170	170
TestArrivalpointWritable		122	122	122	122	120	120
GetExecutionTime		20	20	20	20	22	20
GetLargestExecutionTime		126	126	126	126	126	126
ResetLargestExecutionTime		26	26	26	26	26	26
GetStackOffset		92	92	92	92	90	92

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	296	628	1182	360	598	1332
	NS	276	606	1162	340	578	1308
	KL	244	574	1128	306	544	1274
TerminateTask	LExt	0	0	0	0	0	0
	H	1042	1042	1046	1052	1050	1046
ChainTask	SWL	1734	2066	2984	2070	2238	3326
	SWH	1884	2214	3122	2208	2380	3472
	NSL	1736	2066	2980	2068	2240	3328
	NSH	1802	2134	3040	2128	2368	3456

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Schedule	SW	248	248	358	248	248	360
GetTaskID		154	156	152	154	154	154
GetTaskState		256	256	256	370	370	368
EnableAllInterrupts		26	26	26	26	26	26
DisableAllInterrupts		28	28	28	28	30	28
ResumeAllInterrupts		42	42	42	42	40	42
SuspendAllInterrupts		48	48	48	48	48	48
ResumeOSInterrupts		40	40	40	40	42	40
SuspendOSInterrupts		48	48	48	48	48	48
GetResource	Task	278	276	302	276	278	302
	Combined	200	202	200	202	200	200
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	174	176	174	176	174	174
	Combined	220	220	220	220	220	220
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	426	428	424
	NS	n/a	n/a	n/a	424	426	424
	KL	n/a	n/a	n/a	392	392	390
ClearEvent		n/a	n/a	n/a	120	120	120
GetEvent		n/a	n/a	n/a	152	154	152
WaitEvent	<default>	n/a	n/a	n/a	2024	2026	2396
	fp	n/a	n/a	n/a	2084	2084	2462
GetAlarmBase		338	336	338	336	338	338
GetAlarm		482	480	482	482	480	482
SetRelAlarm		560	560	560	560	560	560
SetAbsAlarm		524	524	524	524	524	524
CancelAlarm		222	222	222	224	224	222
InitCounter		268	266	268	268	266	266
GetCounterValue		304	302	302	302	302	302
osek_tick_alarm	<default>	530	530	530	530	530	530
	KL	478	480	480	478	478	478
osek_incr_counter		162	162	160	160	162	162
GetActiveApplicationMode		12	12	12	12	12	12
StartOS		4194	4194	4194	4194	4194	4194
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	34	32	34	34	32	34
InitCOM		18	18	18	18	18	18
CloseCOM		18	18	18	18	18	18
StartCOM		68	68	68	824	824	824
StopCOM		52	50	52	50	52	52

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ReadFlag		n/a	n/a	n/a	18	18	18
ResetFlag		n/a	n/a	n/a	16	16	16
ReceiveMessage		180	180	180	1588	1588	1624
GetMessageResource		n/a	n/a	n/a	558	558	564
ReleaseMessageResource		n/a	n/a	n/a	414	414	420
GetMessageStatus		n/a	n/a	n/a	350	350	362
SendMessage	SW	632	962	1516	1878	2118	2882
	NS	612	942	1496	1858	2100	2860
	KL	542	872	1426	1790	2030	2792
ActivateTaskset	SW	142	1818	2678	210	1792	2760
	NS	122	1798	2660	190	1772	2744
	KL	74	1764	2632	142	1746	2714
	SW2	142	1818	2678	210	1792	2760
	NS2	122	1798	2658	190	1772	2742
	KL2	76	1766	2630	144	1746	2718
ChainTaskset	SWL	1642	3386	4624	1914	3586	4918
	SWH	1904	3566	4800	2106	3764	5096
	NSL	1652	3396	4628	1924	3594	4926
	NSH	1826	3486	4720	2096	3684	5020
GetTasksetRef		188	188	188	188	188	188
MergeTaskset		300	300	298	300	300	298
AssignTaskset		184	184	184	184	184	184
RemoveTaskset		304	304	306	304	304	306
TestSubTaskset		374	374	372	374	374	372
TestEquivalentTaskset		270	270	270	270	270	270
TickSchedule	SW	1294	3000	3866	1376	3094	4072
	NS	1274	2978	3848	1358	3076	4050
	KL	1240	2944	3810	1322	3040	4016
	SW2	1294	3000	3864	1376	2980	3950
	NS2	1274	2980	3844	1358	2960	3928
	KL2	1238	2944	3810	1322	2926	3892
AdvanceSchedule	SW	756	2460	3330	840	2558	3532
	NS	730	2434	3300	812	2530	3508
	KL	684	2388	3254	766	2484	3460
	SW2	756	2460	3328	838	2442	3410
	NS2	730	2434	3300	812	2414	3384
	KL2	682	2388	3256	766	2370	3340
StartSchedule		388	388	388	388	388	388
StopSchedule		248	248	248	248	248	248
GetScheduleStatus		526	526	526	526	526	526

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetScheduleValue		264	264	264	264	264	264
GetScheduleNext		220	220	220	222	220	220
SetScheduleNext		200	200	202	198	200	200
GetArrivalpointDelay		192	192	192	192	192	192
SetArrivalpointDelay		170	170	168	168	170	170
GetArrivalpointTasksetRef		148	148	150	150	148	148
GetArrivalpointNext		192	192	190	190	192	192
SetArrivalpointNext		168	168	170	170	168	168
TestArrivalpointWritable		122	122	120	120	122	122
GetExecutionTime		374	374	374	374	374	374
GetLargestExecutionTime		268	268	268	270	270	268
ResetLargestExecutionTime		212	212	212	212	212	212
GetStackOffset		92	92	92	92	90	92

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	1786	2122	2656	1842	2080	2806
	NS	2158	2498	3026	2212	2448	3174
	KL	1726	2060	2600	1784	2022	2748
TerminateTask	LExt	1086	1088	1084	1084	1082	1084
	H	1220	1222	1220	1218	1220	1220
ChainTask	SWL	3372	3706	4606	3692	3860	4948
	SWH	3516	3850	4752	3836	4008	5102
	NSL	3754	4088	4988	4078	4314	5402
	NSH	3886	4222	5126	4210	4450	5542
Schedule	SW	362	366	474	364	360	474
GetTaskID		182	184	184	184	182	184
GetTaskState		1780	1782	1782	1860	1856	1858
EnableAllInterrupts		42	40	40	40	38	40
DisableAllInterrupts		42	44	44	44	42	46
ResumeAllInterrupts		56	58	58	58	56	58
SuspendAllInterrupts		60	62	62	62	60	62
ResumeOSInterrupts		56	58	58	60	56	58
SuspendOSInterrupts		60	62	62	62	60	62

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetResource	Task	4438	4362	2020	4832	4830	2412
	Combined	1562	1564	1564	1956	1954	1956
	CLEx	1960	1964	1962	2354	2352	2354
ReleaseResource	Task	1754	1756	1756	2150	2148	2150
	Combined	1498	1500	1500	1892	1890	1892
	CLEx	1668	1674	1670	2064	2060	2062
SetEvent	SW	n/a	n/a	n/a	1904	1904	1902
	NS	n/a	n/a	n/a	2070	2066	2066
	KL	n/a	n/a	n/a	1874	1872	1876
ClearEvent		n/a	n/a	n/a	186	182	182
GetEvent		n/a	n/a	n/a	1782	1780	1782
WaitEvent	<default>	n/a	n/a	n/a	2186	2184	2534
	fp	n/a	n/a	n/a	2248	2246	2600
GetAlarmBase		1358	1360	1364	1360	1358	1360
GetAlarm		1494	1496	1498	1498	1496	1498
SetRelAlarm		1934	1936	1944	1936	1934	1936
SetAbsAlarm		1780	1782	1790	1780	1778	1782
CancelAlarm		1244	1246	1246	1246	1244	1246
InitCounter		1250	1252	1260	1254	1252	1252
GetCounterValue		1182	1186	1188	1184	1184	1184
osek_tick_alarm	<default>	540	542	540	540	538	542
	KL	478	478	478	478	478	478
osek_incr_counter		160	162	162	162	160	160
GetActiveApplicationMode		10	12	12	12	10	12
StartOS		4310	4308	4308	4308	4310	4308
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	38	40	40	40	38	40
InitCOM		16	18	18	18	16	18
CloseCOM		16	18	18	18	16	18
StartCOM		86	88	88	844	842	844
StopCOM		68	70	70	70	68	70
ReadFlag		n/a	n/a	n/a	78	74	76
ResetFlag		n/a	n/a	n/a	60	58	60
ReceiveMessage		808	810	810	2216	2214	2216
GetMessageResource		n/a	n/a	n/a	3154	3152	3154
ReleaseMessageResource		n/a	n/a	n/a	2960	2958	2960
GetMessageStatus		n/a	n/a	n/a	960	958	960
SendMessage	SW	2752	3086	3620	3988	4226	4952
	NS	3122	3462	3990	4358	4596	5318
	KL	2654	2988	3530	3898	4136	4864

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ActivateTaskset	SW	2038	3970	4814	2146	3792	4790
	NS	2326	4262	5106	2434	4082	5082
	KL	1982	3914	4760	2090	3738	4736
	SW2	2038	3968	4816	2144	3792	4790
	NS2	2328	4264	5104	2434	4082	5080
	KL2	1982	3914	4760	2090	3738	4738
ChainTaskset	SWL	4000	5928	7146	4320	5966	7328
	SWH	4182	6112	7324	4502	6152	7510
	NSL	4314	6244	7456	4634	6282	7644
	NSH	4478	6408	7620	4800	6446	7808
GetTasksetRef		1672	1674	1674	1674	1672	1674
MergeTaskset		444	444	444	446	444	446
AssignTaskset		340	342	342	342	340	342
RemoveTaskset		450	452	452	450	448	454
TestSubTaskset		516	518	518	520	518	518
TestEquivalentTaskset		426	428	428	428	426	428
TickSchedule	SW	1824	5296	6142	3472	5438	6424
	NS	2274	5748	6592	3920	5886	6872
	KL	1764	5236	6082	3412	5378	6364
	SW2	1826	5296	6146	3472	5118	6120
	NS2	2274	5748	6590	3920	5566	6564
	KL2	1766	5236	6082	3412	5058	6058
AdvanceSchedule	SW	1350	4818	5666	2994	4960	5946
	NS	1800	5276	6116	3446	5412	6402
	KL	1292	4754	5598	2928	4896	5882
	SW2	1352	4816	5664	2992	4640	5638
	NS2	1800	5274	6116	3446	5092	6090
	KL2	1292	4754	5602	2930	4576	5576
StartSchedule		658	660	664	660	658	660
StopSchedule		302	308	306	304	302	308
GetScheduleStatus		586	588	588	588	586	588
GetScheduleValue		324	326	326	326	324	326
GetScheduleNext		288	294	292	290	288	292
SetScheduleNext		500	502	502	502	500	502
GetArrivalpointDelay		308	310	310	310	308	310
SetArrivalpointDelay		378	380	380	380	378	380
GetArrivalpointTasksetRef		232	234	234	234	232	234
GetArrivalpointNext		274	276	276	276	274	276
SetArrivalpointNext		498	502	502	500	498	502
TestArrivalpointWritable		196	196	196	196	196	200
GetExecutionTime		384	382	382	382	384	382
GetLargestExecutionTime		1688	1690	1692	1690	1688	1690
ResetLargestExecutionTime		1632	1636	1634	1634	1632	1634
GetStackOffset		90	90	90	92	90	92

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	15	15	15	15	15	15
	Cat 2	182	184	182	182	182	182

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	15	15	15	15	15	15
	Cat 2	346	346	346	346	346	346

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	15	15	15	15	15	15
	Cat 2	346	346	346	346	346	346

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()`).

SSX5 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 SSX5 switching contexts measured.

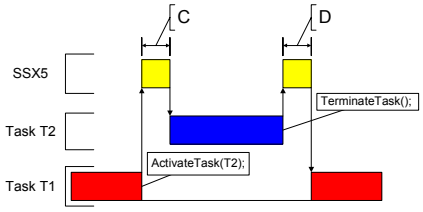


Figure 1: Task Activates a Higher Priority Task which Terminates Normally

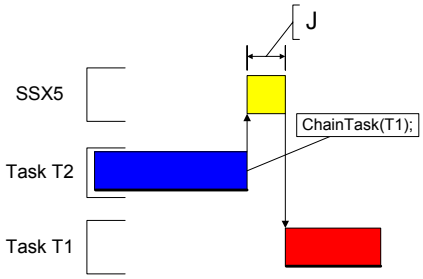


Figure 2: Task Chaining

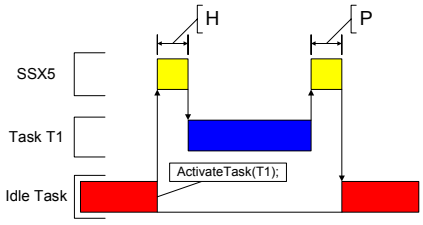


Figure 3: Task Activation from Idle Task

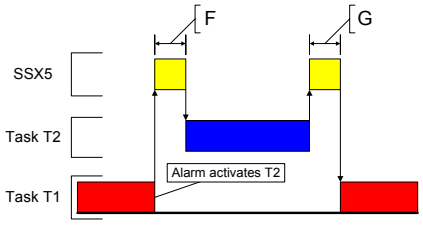


Figure 4: Task Activation from an Alarm

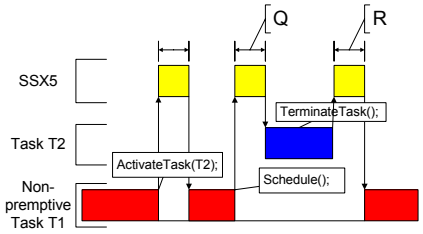


Figure 5: Non-Preemptive Task Calls Schedule()

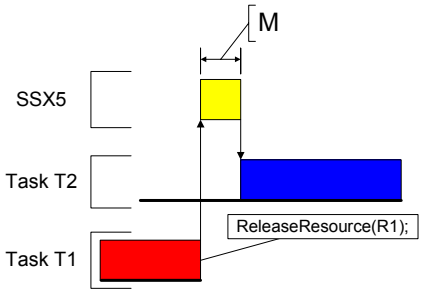


Figure 6: Blocked Task Activated by ReleaseResource()

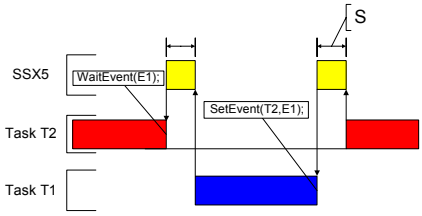


Figure 7: Waiting Task Activated by SetEvent()

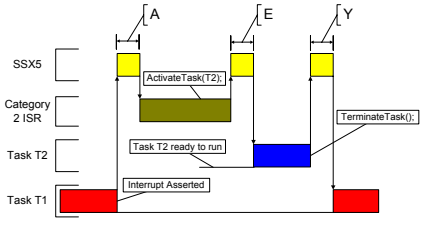


Figure 8: Category 2 ISR Activates a Higher Priority Task

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	222	310	588	230	318	586
Figure 1: D	Heavy, Basic/Extended	350	434	724	548	552	824
ChainTask	Light, Basic	640	1036	1946	702	1050	2118
Figure 2: J	Heavy, Basic/Extended	1186	1666	2854	1438	1790	3128
Pre-emption	Light, Basic	598	996	1910	654	1002	2086
Figure 1: C	Heavy, Basic/Extended	786	1118	2042	1104	1342	2448
From idle task	Light, Basic	598	996	1910	654	1002	2084
Figure 3: H	Heavy, Basic/Extended	786	1116	2038	1102	1342	2444
Triggered by alarm	Light, Basic	1142	1538	2456	1198	1546	2628
Figure 4: F	Heavy, Basic/Extended	1330	1660	2580	1646	1884	2986
Schedule	Light, Basic	466	532	1008	466	532	1008
Figure 5: Q	Heavy, Basic/Extended	654	654	1134	916	916	1394
Release resource	Light, Basic	556	622	988	556	622	988
Figure 6: M	Heavy, Basic/Extended	744	744	1116	1006	1006	1374
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1706	1706	2980
From category 2 ISR	Light, Basic	354	420	784	354	420	784
Figure 8: E	Heavy, Basic/Extended	540	542	912	802	802	1170

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	918	1004	1272	926	1012	1272
Figure 1: D	Heavy, Basic/Extended	1040	1130	1414	1246	1244	1514
ChainTask	Light, Basic	1608	1946	2858	1672	1960	3028
Figure 2: J	Heavy, Basic/Extended	2908	3336	4518	3100	3390	4722
Pre-emption	Light, Basic	1390	1728	2642	1446	1734	2814
Figure 1: C	Heavy, Basic/Extended	1518	1850	2770	1844	2084	3184

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
From idle task	Light, Basic	1394	1728	2638	1446	1734	2812
Figure 3: H	Heavy, Basic/Extended	1520	1848	2766	1844	2082	3180
Triggered by alarm	Light, Basic	1934	2270	3188	1990	2278	3356
Figure 4: F	Heavy, Basic/Extended	2062	2392	3312	2386	2626	3724
Schedule	Light, Basic	1258	1264	1736	1258	1264	1736
Figure 5: Q	Heavy, Basic/Extended	1386	1386	1862	1656	1656	2132
Release resource	Light, Basic	1348	1354	1716	1348	1354	1716
Figure 6: M	Heavy, Basic/Extended	1476	1476	1844	1746	1748	2110
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	2436	2434	3706
From category 2 ISR	Light, Basic	2102	2108	2470	2102	2108	2470
Figure 8: E	Heavy, Basic/Extended	2230	2230	2596	2502	2500	2864

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	1086	1176	1436	1084	1170	1436
Figure 1: D	Heavy, Basic/Extended	1220	1382	1588	1416	1412	1688
ChainTask	Light, Basic	3240	3580	4472	3288	3574	4642
Figure 2: J	Heavy, Basic/Extended	4716	5148	6316	4890	5248	6590
Pre-emption	Light, Basic	2860	3202	4100	2920	3206	4276
Figure 1: C	Heavy, Basic/Extended	2994	3330	4238	3324	3562	4652
From idle task	Light, Basic	2866	3202	4100	2920	3212	4274
Figure 3: H	Heavy, Basic/Extended	3000	3328	4234	3324	3566	4652
Triggered by alarm	Light, Basic	3414	3756	4658	3474	3764	4828
Figure 4: F	Heavy, Basic/Extended	3548	3882	4788	3878	4114	5202
Schedule	Light, Basic	1350	1358	1834	1352	1356	1830
Figure 5: Q	Heavy, Basic/Extended	1482	1484	1964	1756	1754	2230

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Release resource	Light, Basic	2840	2848	3210	3234	3238	3602
Figure 6: M	Heavy, Basic/Extended	2974	2976	3344	3638	3638	4002
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	3964	3964	5222
From category 2 ISR	Light, Basic	2108	2116	2478	2110	2114	2478
Figure 8: E	Heavy, Basic/Extended	2242	2244	2610	2514	2512	2878

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. RTArchitect 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	11	11	13	11	11	13
BCC1 lightweight, floating-point	13	13	15	13	13	15
BCC1 heavyweight, integer	11	11	13	11	11	13
BCC1 heavyweight, floating-point	11	11	13	11	11	13
BCC2 lightweight, integer	n/a	13	17	n/a	13	17
BCC2 lightweight, floating-point	n/a	13	17	n/a	13	17
BCC2 heavyweight, integer	n/a	11	17	n/a	11	17
BCC2 heavyweight, floating-point	n/a	11	17	n/a	11	17
ECC1 heavyweight, integer	n/a	n/a	n/a	12	12	14
ECC1 heavyweight, floating-point	n/a	n/a	n/a	12	12	14
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	17
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	17

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		11	11	13	11	11	13
BCC1 lightweight, floating-point		13	13	15	13	13	15
BCC1 heavyweight, integer		11	11	13	11	11	13
BCC1 heavyweight, floating-point		11	11	13	11	11	13
BCC2 lightweight, integer		n/a	13	17	n/a	13	17
BCC2 lightweight, floating-point		n/a	13	17	n/a	13	17
BCC2 heavyweight, integer		n/a	11	17	n/a	11	17
BCC2 heavyweight, floating-point		n/a	11	17	n/a	11	17
ECC1 heavyweight, integer		n/a	n/a	n/a	12	12	14
ECC1 heavyweight, floating-point		n/a	n/a	n/a	12	12	14
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	17
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	17

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		17	17	17	17	17	17
BCC1 lightweight, floating-point		19	19	19	19	19	19
BCC1 heavyweight, integer		17	17	17	17	17	17
BCC1 heavyweight, floating-point		17	17	17	17	17	17
BCC2 lightweight, integer		n/a	19	21	n/a	19	21
BCC2 lightweight, floating-point		n/a	19	21	n/a	19	21
BCC2 heavyweight, integer		n/a	17	21	n/a	17	21
BCC2 heavyweight, floating-point		n/a	17	21	n/a	17	21
ECC1 heavyweight, integer		n/a	n/a	n/a	18	18	18
ECC1 heavyweight, floating-point		n/a	n/a	n/a	18	18	18
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	21
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	21
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		17	17	17	17	17	17
BCC1 lightweight, floating-point		19	19	19	19	19	19
BCC1 heavyweight, integer		17	17	17	17	17	17
BCC1 heavyweight, floating-point		17	17	17	17	17	17
BCC2 lightweight, integer		n/a	19	21	n/a	19	21

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC2 lightweight, floating-point		n/a	19	21	n/a	19	21
BCC2 heavyweight, integer		n/a	17	21	n/a	17	21
BCC2 heavyweight, floating-point		n/a	17	21	n/a	17	21
ECC1 heavyweight, integer		n/a	n/a	n/a	18	18	18
ECC1 heavyweight, floating-point		n/a	n/a	n/a	18	18	18
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	21
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	21

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		17	17	17	17	17	17
BCC1 lightweight, floating-point		19	19	19	19	19	19
BCC1 heavyweight, integer		17	17	17	17	17	17
BCC1 heavyweight, floating-point		17	17	17	17	17	17
BCC2 lightweight, integer		n/a	19	21	n/a	19	21
BCC2 lightweight, floating-point		n/a	19	21	n/a	19	21
BCC2 heavyweight, integer		n/a	17	21	n/a	17	21
BCC2 heavyweight, floating-point		n/a	17	21	n/a	17	21
ECC1 heavyweight, integer		n/a	n/a	n/a	18	18	18
ECC1 heavyweight, floating-point		n/a	n/a	n/a	18	18	18
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	21
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	21
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		17	17	17	17	17	17
BCC1 lightweight, floating-point		19	19	19	19	19	19
BCC1 heavyweight, integer		17	17	17	17	17	17
BCC1 heavyweight, floating-point		17	17	17	17	17	17
BCC2 lightweight, integer		n/a	19	21	n/a	19	21
BCC2 lightweight, floating-point		n/a	19	21	n/a	19	21
BCC2 heavyweight, integer		n/a	17	21	n/a	17	21
BCC2 heavyweight, floating-point		n/a	17	21	n/a	17	21
ECC1 heavyweight, integer		n/a	n/a	n/a	18	18	18
ECC1 heavyweight, floating-point		n/a	n/a	n/a	18	18	18
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	21
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	21

5 Use of @stack and @nostack

The OS header files enforce a stack-based memory model for all OSEK-defined function types. With the appropriate safeguards, however, you are free to choose between a stack-based memory model (e.g. function modifier `@stack`) and a memory-based model (e.g. function modifier `@nostack`) for the remaining functions in the application.

Memory model functions share an overlay area of static RAM and, to ensure correct multi-threading behavior, it is usually necessary to treat it as follows:

1. Declare to RTArchitect a standard resource "`nostack`" (the actual name is unimportant).
2. Declare to RTArchitect each task that calls any `@nostack` functions as using resource "`nostack`".
3. Protect each call to a `@nostack` function by locking the resource "`nostack`", e.g.

```
GetResource_nostack();
nostack_function();
ReleaseResource_nostack();
```

This resource lock is necessary due to the call-graph analysis performed by the Cosmic linker. To minimize RAM usage, the linker identifies independent `@nostack` memory that may be overlaid into a common area of RAM. The linker, however, is not necessarily aware of the possible interactions between task bodies. It cannot take into account the task preemption that can occur with SSX5.

Consider an example of two tasks, A and B. Task A is at a higher priority than task B and both tasks have a `@nostack` worker function, called `support_A()` and `support_B()` respectively.

The linker may identify that `support_A()` and `support_B()` are independent and, therefore, re-use the same area of RAM for both routines' locals. If an interrupt occurs inside `support_B()`, and if the ISR activates task A, task A will be set *running* as soon as the ISR terminates (see the *RTA User Guide* for a definition of the term '*running*'). Any locals belonging to `support_B()` will become corrupted when the task A worker function is invoked.

Note that Category 1 ISRs may call `@nostack` functions without resource protection. Category 1 ISRs must be declared with the `@interrupt` modifier, as this ensures that the processor registers and compiler pseudo-registers are preserved. The compiler includes this information in the object file. The linker then ensures that memory for Category 1 `@nostack` functions does not overlay that used at user level.

Support Details

Getting Help

There are a number of ways to contact LiveDevices for technical support. When you contact our support team, please provide your customer number.

Email

The preferred method for dealing with support inquiries is via email. Any issues should be sent to **support@livedevices.com**

Telephone

You can contact us by telephone during our normal office hours (0900-1730 GMT/BST). Our telephone number is +44 (0) 19 04 56 26 24

Fax

Our Fax number is +44 (0) 19 04 56 25 81

World Wide Web

You can keep up with the latest developments by looking at our web site **www.livedevices.com**

Write to Us

You can write to us at:

LiveDevices Ltd.
Atlas House
Link Business Park
Osbaldwick Link Road
Osbaldwick
York
YO10 3JB