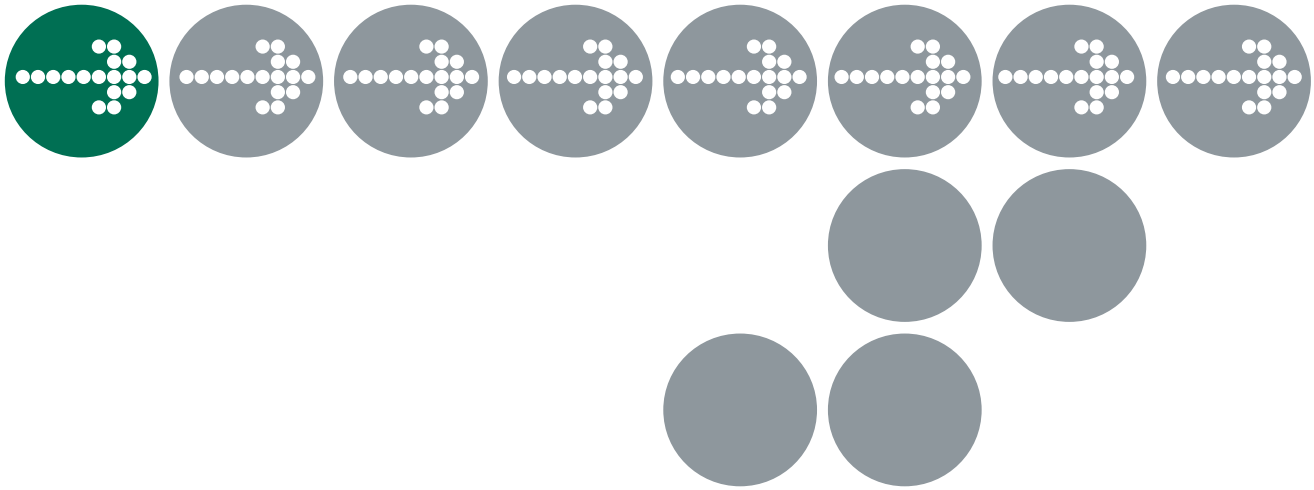
LiveDevices

Binding Manual: ST30/ADS



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: RM00008-002 .03

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

This guide provides port specific information for the ST30/ADS 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 Compiler

SSX5 was built using the following compiler:

Vendor	ARM
Compiler	ARM Developer Suite
Version	ADS v1.2

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

Option	Description
`-li`	Compile for little-endian byte access format.

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.

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 SSX5 run-time libraries have mainly been compiled using the THUMB instruction set.

2.2 Assembler

SSX5 was built using the following assembler:

Vendor	ARM
Assembler	ARM/THUMB Macro Assembler
Version	ADS v1.2

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

Option	Description
`-li`	Compile for little-endian byte access format.

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.

2.3 Linker/Locator

The compulsory linker/locator options for an SSX5 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 sections must be located:

Sections	ROM/RAM	Description
`os_pid`	ROM	SSX5 read-only data.
`os_pird`	ROM	SSX5 initialization data.
`os_vectbl`	ROM	Vector table if generated by RTArchitect.
`os_pir`	RAM	SSX5 initialized data.
`os_pur`	RAM	SSX5 uninitialized data.

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 to start at address 0x0.

The following compiler run-time library functions are required by SSX5:

C Library Functions	Description
`setjmp`	ARM instruction `setjmp` routine.
`_16setjmp`	THUMB instruction `setjmp` routine.
`longjmp`	ARM instruction `longjmp` routine.
`_16longjmp`	THUMB instruction `longjmp` routine.
`callvia`	Indirect function call.

2.3.1 Linker Command File

The RAM sections that are dedicated to RTA consist of the `os_pur` and `os_pir` sections. These sections are initialized by RTA within the `StartOS()` API call (unlike standard C RAM variable sections, which are initialized in the application start-up code). These sections should be marked `UNINIT` in the linker command file, which prevents the contents of the section being additionally initialized in the application start-up code.

2.4 Debugger

ORTI is the OSEK Run-Time Interface. RTA does not currently support ORTI compatible debuggers for this target.

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 *ST30 Target specification*.

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

IPL Values	CIPR [3:0]	CPSR [7:6]	Description
0	0000	00	User Level.
1	0001	00	OS Level - IRQ Category 2 interrupt level. Category 2 interrupts share the same level, so the priority relationship is dependent upon the EIC arbitration order.
2 - 15	0010 – 1111	00	IRQ Category 1 interrupt levels.
16	N/A	11	Non IRQ Category 1 interrupt level.

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
0x04	Undefined Instructions
0x08	SWI
0x0C	Prefetch Abort
0x10	Data Abort
0x14	Reserved
0x18	IRQ - Automatically reserved by RTA see Section 3.1.6
0x1C	FIQ (Fast Interrupt)
0x20	IRQ 00 - see Section 3.1.7
0x24	IRQ 01
0x28 - - 0x118	IRQ 02 - - IRQ 62
0x11C	IRQ 63

The valid base addresses for the vector table are:

Base Address	Notes
0x0	

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

Vector Location	Label
0xVVV	_os_wrapper_VVV
e.g. 0x064	_os_wrapper_064

3.1.6 ARM CPU IRQ Vector

In order to support multi-entry Enhanced Interrupt Controller (EIC) interrupt vector operation, the ARM CPU IRQ vector at address 0x18 must always reference the EIC Interrupt Vector Register (IVR). RTA automatically configures the IRQ vector to reference the address held by IVR, as this holds the address of the exception handler when an IRQ exception is generated.

If you are interested in using a single entry point IRQ interrupt scheme you should contact LiveDevices to confirm whether or not this is possible.

3.1.7 Software Vector Table (IRQ 00 to IRQ 63)

The vector table used in the ST30/ADS implementation of RTA concatenates the ARM CPU vector table with the 64 interrupt channels of the (EIC) (see Section 3.1.2). The offset addresses, shown in Section 3.1.2, illustrate the direct mapping to the interrupt source that should be used in the OIL configuration file.

3.1.8 Initializing the EIC

The EIC IVR and channel Source Interrupt Registers (SIRs) require initialization in all applications that contain IRQ interrupts. RTA supports the initialization of these registers, using details from the OIL configuration files, in the following ways:

IVR Setup

The label `os_ivr_setup` is generated in the file `osgen.s` at the start of the Category 2 interrupt handler functions. The address of this label should be used to initialize the IVR.

The read-only code section of `osgen.s` should be located first, followed by the other read-only code sections containing the entries to the other Category 1 IRQ handler functions. The upper 2-bytes of these sections in memory must be common.

The IVR is automatically updated by the EIC to hold the address of the IRQ interrupt handler when an IRQ interrupt occurs. The lower 2-bytes are taken from the SIR of the interrupt to be serviced, whilst the upper 2-bytes held in the IVR are unchanged. The upper 2-bytes are common to the entry address of all IRQ exception handlers, so they must reside within the same 64-Kbyte block of memory. It should be noted that this address restriction does not apply to Category 2 ISRs because they are standard C functions. The entry points of the Category 2 interrupt handler functions reside in `osgen.s`.

SIR Setup

RTA provides two arrays `os_irq_vectors` and `os_irq_ipr_values` in the file `osgen.s`. They can be used to configure the SIRs. The array `os_irq_vectors` contains the 4-byte addresses of the entry function and the array `os_irq_ipr_values` contains the priority levels for the 64 EIC channels. A suggested method to initialize the SIRs using these arrays can be found in the example application.

3.1.9 Reset Vector

In the generated vector table, the Reset Vector at 0x0 is always attached to a user function `reset_handler()`. This reset handler function should perform the start-up operation required for the ARM C compiler and setup the stack pointers for the CPU modes used. An example of `reset_handler()` is provided in the example application.

3.2 CPU Operating Modes

An ST30 CPU can use up to six different modes in an application, each with their own stack. SSX5 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, SSX5 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 SSX5, 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.

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, `f_link.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.

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

3.3 Register Settings

SSX5 requires the following registers to be initialized before calling `StartOS()`.

Register	Required Value
IVR (Interrupt Vector Register)	Top 2-bytes common to the IRQ exception handler's entry address.
SIRn (Source Interrupt Registers)	Lower 2-bytes of the entry address and the priority level of IRQ exception handler 'n'.

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

Registers Used	Notes
CPSR	Used to control the I and F bits and CPU mode.
CIPR	Used to set the priority level in the EIC.

3.4 Stack Usage

3.4.1 Number of Stacks

Two stacks are used. The SVC stack indexed 0 and the IRQ stack indexed 1. The IRQ stack is only used on entry to Category 2 interrupts and is not manipulated in any SSX5 API calls. The first argument to `StackFaultHook` is always 0 as excess will only be measured by SSX5 on the SVC stack.

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

3.4.2 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): 52

Timing

API max usage (bytes): 52

Extended

API max usage (bytes): 84

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.

3.5 Converting Configuration Files from the ST ARGUS Port of RTA

An earlier version of the ST30/ADS implementation of RTA was released before the ST30 numbering scheme was developed. This used the processor name ST ARGUS.

The OIL configuration files of the ARGUS/ARMADS implementation of RTA should be modified for use in the ST30/ADS implementation of RTA. To convert an OIL configuration file, the following lines should be modified in a text editor:

```
//RTAOILCFG OS_VERSION "v3.0";
//RTAOILCFG OS_TARGET "ARGUS/ARMADS";
```

should be changed to:

```
//RTAOILCFG OS_VERSION "v3.1";
//RTAOILCFG OS_TARGET "ST30/ADS";
```

Once the OIL configuration file has been saved, it can be opened in RTA if any further modifications are required.

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	**ST ARGUS**
Clock speed (MHz)	14
Code memory	On-chip RAM
Read-only data memory	On-chip RAM
Read-write data memory	On-chip RAM

4.1 Functionality

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

Configuration	**Application Uses**					
Events	**No**			**Yes**		
Shared Task Priorities	**No**		**Yes**	**No**		**Yes**
Multiple Task Activations	**No**	**Yes**		**No**	**Yes**	
Maximum number of tasks	32	32	32	32	32	32
Maximum number of not suspended tasks	32	32	32	32	32	32
Maximum number of priorities	32	32	32	32	32	32
Number of tasks per priority (for BCC2 and ECC2)	n/a	32	32	n/a	32	32
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	32	32	32
Limits for the number of alarm objects (per system / per task)	not limited by 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	24	24	24	24	24	24
	ROM	144	144	144	144	144	144
COM overhead	RAM	2	2	2	2	2	2
	ROM	9	9	9	9	9	9

Timing

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	36	36	36	36	36	36
	ROM	184	184	184	184	184	184
COM overhead	RAM	2	2	2	2	2	2
	ROM	9	9	9	9	9	9

Extended

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
OS overhead	RAM	48	48	48	48	48	48
	ROM	222	222	222	222	222	222
COM overhead	RAM	2	2	2	2	2	2
	ROM	9	9	9	9	9	9

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	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	52	52	52	52	52	52
Category 2 ISR, floating point	RAM	1	1	1	1	1	1
	ROM	72	72	72	72	72	72

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
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	6	6	6	6	6	6
	ROM	32	32	32	32	32	32
Counter	RAM	2	2	2	2	2	2
	ROM	44	44	44	44	44	44
Message	RAM	11	11	11	31	31	31
	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	12	12	12	12	12	12
	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	6	6	6	6	6	6
	ROM	44	44	44	44	44	44
BCC1 Heavyweight task	RAM	10	10	10	10	10	10
	ROM	48	48	48	48	48	48
BCC2 task	RAM	n/a	14	16	n/a	14	16
	ROM	n/a	52	60	n/a	52	60
ECC1, Integer task	RAM	n/a	n/a	n/a	146	146	146
	ROM	n/a	n/a	n/a	68	68	68
ECC1, floating point task	RAM	n/a	n/a	n/a	148	148	148
	ROM	n/a	n/a	n/a	68	68	68
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	148
	ROM	n/a	n/a	n/a	n/a	n/a	76
ECC2, floating point task	RAM	n/a	n/a	n/a	n/a	n/a	150
	ROM	n/a	n/a	n/a	n/a	n/a	76
Category 2 ISR	RAM	6	6	6	6	6	6
	ROM	86	86	86	86	86	86
Category 2 ISR, floating point	RAM	7	7	7	7	7	7
	ROM	94	94	94	94	94	94
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	6	6	6	6	6	6
	ROM	32	32	32	32	32	32
Counter	RAM	2	2	2	2	2	2
	ROM	44	44	44	44	44	44
Message	RAM	11	11	11	31	31	31
	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

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	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	12	12	12	12	12	12
	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	7	7	7	7	7	7
	ROM	52	52	52	52	52	52
BCC1 Heavyweight task	RAM	14	14	14	14	14	14
	ROM	52	52	52	52	52	52
BCC2 task	RAM	n/a	18	20	n/a	18	20
	ROM	n/a	56	64	n/a	56	64
ECC1, Integer task	RAM	n/a	n/a	n/a	150	150	150
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating point task	RAM	n/a	n/a	n/a	152	152	152
	ROM	n/a	n/a	n/a	72	72	72
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	152
	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	154
	ROM	n/a	n/a	n/a	n/a	n/a	80
Category 2 ISR	RAM	7	7	7	7	7	7
	ROM	94	94	94	94	94	94
Category 2 ISR, floating point	RAM	8	8	8	8	8	8
	ROM	102	102	102	102	102	102
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

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Linked resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Alarm	RAM	6	6	6	6	6	6
	ROM	36	36	36	36	36	36
Counter	RAM	2	2	2	2	2	2
	ROM	48	48	48	48	48	48
Message	RAM	11	11	11	31	31	31
	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
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	18	18	18	18	18	18
Arrivalpoint (writable)	RAM	18	18	18	18	18	18
	ROM	18	18	18	18	18	18
Schedule	RAM	14	14	14	14	14	14
	ROM	42	42	42	42	42	42
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

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, module 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	90	146	178	102	154	202
	NS		72	128	160	84	136	184
	KL	2	50	102	134	62	114	158
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	26	26	26	26	26	26
ChainTask	SWL	1, 8	84	136	164	96	144	196
	SWH	1, 9	110	166	198	118	174	222
	NSL	8	84	136	164	96	144	196
	NSH	9	98	154	186	106	162	210

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
Schedule			74	74	98	74	74	98
GetTaskID			28	28	28	28	28	28
GetTaskState			72	72	72	88	88	88
EnableAllInterrupts			46	46	46	46	46	46
DisableAllInterrupts			38	38	38	38	38	38
ResumeAllInterrupts			62	62	62	62	62	62
SuspendAllInterrupts			50	50	50	50	50	50
ResumeOSInterrupts			48	48	48	48	48	48
SuspendOSInterrupts			52	52	52	52	52	52
GetResource	Task	7	24	24	28	24	24	28
	Combined	6	62	62	62	62	62	62
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	58	58	58	58	58	58
	Combined	6	94	94	94	94	94	94
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	102	102	174
	NS		n/a	n/a	n/a	84	84	156
	NS1i	10	n/a	n/a	n/a	48	n/a	n/a
	KL	2	n/a	n/a	n/a	60	60	132
	KL1i	2, 10	n/a	n/a	n/a	24	n/a	n/a
ClearEvent			n/a	n/a	n/a	42	42	42
GetEvent			n/a	n/a	n/a	36	36	36
WaitEvent	<default>		n/a	n/a	n/a	152	152	270
	fp	11	n/a	n/a	n/a	168	168	306
	1i	10	n/a	n/a	n/a	20	n/a	n/a
GetAlarmBase			42	42	42	42	42	42
GetAlarm			70	70	70	70	70	70
SetRelAlarm			78	78	78	78	78	78
SetAbsAlarm			94	94	94	94	94	94
CancelAlarm			58	58	58	58	58	58
InitCounter			52	52	52	52	52	52
GetCounterValue			60	60	60	60	60	60
osek_tick_alarm	<default>		60	60	60	60	60	60
	KL	2	32	32	32	32	32	32
osek_incr_counter			32	32	32	32	32	32
GetActiveApplicationMode		31	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			124	124	124	124	124	124
ShutdownOS	NoHook	12	24	24	24	24	24	24
	Hook	13	30	30	30	30	30	30
InitCOM			4	4	4	4	4	4

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
CloseCOM			4	4	4	4	4	4
StartCOM			20	20	20	20	20	20
StopCOM			20	20	20	20	20	20
ReadFlag		31	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		31	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	46	46	46	128	128	128
	CCCB	15	128	128	128	128	128	128
GetMessageResource			42	42	42	42	42	42
ReleaseMessageResource			38	38	38	38	38	38
GetMessageStatus			40	40	40	40	40	40
SendMessage	SW CCCA	1, 14	62	62	62	158	158	158
	SW CCCB	1, 15	148	148	148	158	158	158
	NS CCCA	14	62	62	62	158	158	158
	NS CCCB	15	148	148	148	158	158	158
	KL CCCA	2, 14	42	42	42	138	138	138
	KL CCCB	2, 15	128	128	128	138	138	138
main_dispatch	NoHook	12	90	90	122	90	90	122
	Hook	13	118	118	150	118	118	150
sub_dispatch	B1LI	19	22	22	22	22	22	22
	B1LF	20	26	26	26	26	26	26
	B1HI	21	66	66	66	66	66	66
	B1HF	22	70	70	70	70	70	70
	B2LI	23	n/a	60	80	n/a	60	80
	B2LF	24	n/a	64	84	n/a	64	84
	B2HI	25	n/a	102	154	n/a	102	154
	B2HF	26	n/a	106	158	n/a	106	158
	E1HI	27	n/a	n/a	n/a	216	216	268
	E1HF	28	n/a	n/a	n/a	220	220	272
	E2HI	29	n/a	n/a	n/a	n/a	n/a	268
	E2HF	30	n/a	n/a	n/a	n/a	n/a	272
ErrorHook support		16	34	34	34	34	34	34
	ServiceID	17	42	42	42	42	42	42
	Parameters	18	50	50	50	50	50	50
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	62	106	142	70	126	170
	NS		44	88	124	52	108	152
	KL	2	20	68	104	28	88	128

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
ChainTaskset	SWL	1, 8	46	90	126	46	102	142
	SWH	1, 9	80	128	164	80	136	180
	NSL	8	46	90	126	46	102	142
	NSH	9	68	116	152	68	124	168
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			40	40	40	40	40	40
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			40	40	40	40	40	40
TestSubTaskset			48	48	48	48	48	48
TestEquivalentTaskset			48	48	48	48	48	48
TickSchedule	SW	1	126	120	120	120	120	120
	NS		104	94	94	94	94	94
	KL	2	84	78	78	78	78	78
AdvanceSchedule	SW	1	108	102	102	102	102	102
	NS		86	76	76	76	76	76
	KL	2	66	60	60	60	60	60
StartSchedule			62	62	62	62	62	62
StopSchedule			46	46	46	46	46	46
GetScheduleStatus			74	74	74	74	74	74
GetScheduleValue			50	50	50	50	50	50
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			8	8	8	8	8	8
SetArrivalpointNext			8	8	8	8	8	8
TestArrivalpointWritable			32	32	32	32	32	32
GetExecutionTime			4	4	4	4	4	4
GetLargestExecutionTime			8	8	8	8	8	8
ResetLargestExecutionTime			4	4	4	4	4	4
GetStackOffset			26	26	26	26	26	26
Interrupt wrapper			136	136	136	136	136	136
Support routines			48	48	48	48	48	48
Support routines			22	22	22	22	22	22
Support routines			98	98	98	98	98	98
Support routines			18	18	18	18	18	18
Support routines			78	78	78	78	78	78
Support routines			8	8	8	8	8	8
Support routines			20	20	20	20	20	20

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	90	146	178	102	154	202
	NS		72	128	160	84	136	184
	KL	2	50	102	134	62	114	158
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	26	26	26	26	26	26
ChainTask	SWL	1, 8	84	136	164	96	144	196
	SWH	1, 9	110	166	198	118	174	222
	NSL	8	84	136	164	96	144	196
	NSH	9	98	154	186	106	162	210
Schedule			94	94	118	94	94	118
GetTaskID			28	28	28	28	28	28
GetTaskState			72	72	72	88	88	88
EnableAllInterrupts			46	46	46	46	46	46
DisableAllInterrupts			38	38	38	38	38	38
ResumeAllInterrupts			62	62	62	62	62	62
SuspendAllInterrupts			50	50	50	50	50	50
ResumeOSInterrupts			48	48	48	48	48	48
SuspendOSInterrupts			52	52	52	52	52	52
GetResource	Task	7	24	24	28	24	24	28
	Combined	6	62	62	62	62	62	62
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	78	78	78	78	78	78
	Combined	6	130	130	130	130	130	130
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	102	102	174
	NS		n/a	n/a	n/a	84	84	156
	NS1i	10	n/a	n/a	n/a	48	n/a	n/a
	KL	2	n/a	n/a	n/a	60	60	132
	KL1i	2, 10	n/a	n/a	n/a	24	n/a	n/a
ClearEvent			n/a	n/a	n/a	42	42	42
GetEvent			n/a	n/a	n/a	36	36	36
WaitEvent	<default>		n/a	n/a	n/a	152	152	270
	fp	11	n/a	n/a	n/a	168	168	306
	1i	10	n/a	n/a	n/a	20	n/a	n/a
GetAlarmBase			42	42	42	42	42	42
GetAlarm			70	70	70	70	70	70
SetRelAlarm			78	78	78	78	78	78
SetAbsAlarm			94	94	94	94	94	94
CancelAlarm			58	58	58	58	58	58

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
InitCounter			52	52	52	52	52	52
GetCounterValue			60	60	60	60	60	60
osek_tick_alarm	<default>		60	60	60	60	60	60
	KL	2	32	32	32	32	32	32
osek_incr_counter			32	32	32	32	32	32
GetActiveApplicationMode		31	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			164	164	164	164	164	164
ShutdownOS	NoHook	12	24	24	24	24	24	24
	Hook	13	30	30	30	30	30	30
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			20	20	20	20	20	20
StopCOM			20	20	20	20	20	20
ReadFlag		31	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		31	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	46	46	46	128	128	128
	CCCB	15	128	128	128	128	128	128
GetMessageResource			42	42	42	42	42	42
ReleaseMessageResource			38	38	38	38	38	38
GetMessageStatus			40	40	40	40	40	40
SendMessage	SW CCCA	1, 14	62	62	62	158	158	158
	SW CCCB	1, 15	148	148	148	158	158	158
	NS CCCA	14	62	62	62	158	158	158
	NS CCCB	15	148	148	148	158	158	158
	KL CCCA	2, 14	42	42	42	138	138	138
	KL CCCB	2, 15	128	128	128	138	138	138
main_dispatch	NoHook	12	144	144	176	144	144	176
	Hook	13	176	176	208	176	176	208
sub_dispatch	B1LI	19	10	10	10	10	10	10
	B1LF	20	14	14	14	14	14	14
	B1HI	21	68	68	68	68	68	68
	B1HF	22	72	72	72	72	72	72
	B2LI	23	n/a	50	70	n/a	50	70
	B2LF	24	n/a	54	74	n/a	54	74
	B2HI	25	n/a	96	148	n/a	96	148
	B2HF	26	n/a	100	152	n/a	100	152
	E1HI	27	n/a	n/a	n/a	238	238	290
	E1HF	28	n/a	n/a	n/a	242	242	294
	E2HI	29	n/a	n/a	n/a	n/a	n/a	290
	E2HF	30	n/a	n/a	n/a	n/a	n/a	294

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
ErrorHook support		16	34	34	34	34	34	34
	ServiceID	17	42	42	42	42	42	42
	Parameters	18	50	50	50	50	50	50
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	66	66	66	66	66	66
Timing_termination		4	68	68	68	68	68	68
ActivateTaskset	SW	1	62	106	142	70	126	170
	NS		44	88	124	52	108	152
	KL	2	20	68	104	28	88	128
ChainTaskset	SWL	1, 8	46	90	126	46	102	142
	SWH	1, 9	80	128	164	80	136	180
	NSL	8	46	90	126	46	102	142
	NSH	9	68	116	152	68	124	168
GetTasksetRef			8	8	8	8	8	8
MergeTaskset			40	40	40	40	40	40
AssignTaskset			8	8	8	8	8	8
RemoveTaskset			40	40	40	40	40	40
TestSubTaskset			48	48	48	48	48	48
TestEquivalentTaskset			48	48	48	48	48	48
TickSchedule	SW	1	126	120	120	120	120	120
	NS		104	94	94	94	94	94
	KL	2	84	78	78	78	78	78
AdvanceSchedule	SW	1	108	102	102	102	102	102
	NS		86	76	76	76	76	76
	KL	2	66	60	60	60	60	60
StartSchedule			62	62	62	62	62	62
StopSchedule			46	46	46	46	46	46
GetScheduleStatus			74	74	74	74	74	74
GetScheduleValue			50	50	50	50	50	50
GetScheduleNext			12	12	12	12	12	12
SetScheduleNext			8	8	8	8	8	8
GetArrivalpointDelay			8	8	8	8	8	8
SetArrivalpointDelay			8	8	8	8	8	8
GetArrivalpointTasksetRef			8	8	8	8	8	8
GetArrivalpointNext			8	8	8	8	8	8
SetArrivalpointNext			8	8	8	8	8	8
TestArrivalpointWritable			32	32	32	32	32	32
GetExecutionTime			86	86	86	86	86	86
GetLargestExecutionTime			12	12	12	12	12	12
ResetLargestExecutionTime			12	12	12	12	12	12

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetStackOffset			26	26	26	26	26	26
Interrupt wrapper			136	136	136	136	136	136
Support routines			48	48	48	48	48	48
Support routines			22	22	22	22	22	22
Support routines			98	98	98	98	98	98
Support routines			18	18	18	18	18	18
Support routines			78	78	78	78	78	78
Support routines			8	8	8	8	8	8
Support routines			20	20	20	20	20	20

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	172	228	260	184	236	284
	NS		214	274	302	226	282	326
	KL	2	130	186	218	142	194	242
TerminateTask	LExt	3	102	102	102	102	102	102
	H	5	116	116	116	116	116	116
ChainTask	SWL	1, 8	208	260	288	220	268	320
	SWH	1, 9	234	294	326	246	306	350
	NSL	8	260	312	344	272	324	376
	NSH	9	278	338	370	290	350	394
Schedule			174	174	198	174	174	198
GetTaskID			44	44	44	44	44	44
GetTaskState			172	172	172	180	180	180
EnableAllInterrupts			58	58	58	58	58	58
DisableAllInterrupts			50	50	50	50	50	50
ResumeAllInterrupts			96	96	96	96	96	96
SuspendAllInterrupts			62	62	62	62	62	62
ResumeOSInterrupts			86	86	86	86	86	86
SuspendOSInterrupts			64	64	64	64	64	64
GetResource	Task	7	230	230	206	230	230	206
	Combined	6	214	214	214	214	214	214
	CLEx	3	200	200	200	200	200	200
ReleaseResource	Task	7	232	232	232	232	232	232
	Combined	6	280	280	280	280	280	280
	CLEx	3	190	190	190	190	190	190

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	216	216	292
	NS		n/a	n/a	n/a	262	262	334
	NS1i	10	n/a	n/a	n/a	154	n/a	n/a
	KL	2	n/a	n/a	n/a	174	174	246
	KL1i	2, 10	n/a	n/a	n/a	130	n/a	n/a
ClearEvent			n/a	n/a	n/a	116	116	116
GetEvent			n/a	n/a	n/a	170	170	170
WaitEvent	<default>		n/a	n/a	n/a	226	226	332
	fp	11	n/a	n/a	n/a	242	242	368
	1i	10	n/a	n/a	n/a	110	n/a	n/a
GetAlarmBase			112	112	112	112	112	112
GetAlarm			120	120	120	120	120	120
SetRelAlarm			156	156	156	156	156	156
SetAbsAlarm			172	172	172	172	172	172
CancelAlarm			108	108	108	108	108	108
InitCounter			124	124	124	124	124	124
GetCounterValue			130	130	130	130	130	130
osek_tick_alarm	<default>		76	76	76	76	76	76
	KL	2	32	32	32	32	32	32
osek_incr_counter			32	32	32	32	32	32
GetActiveApplicationMode		31	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			180	180	180	180	180	180
ShutdownOS	NoHook	12	32	32	32	32	32	32
	Hook	13	38	38	38	38	38	38
InitCOM			4	4	4	4	4	4
CloseCOM			4	4	4	4	4	4
StartCOM			32	32	32	32	32	32
StopCOM			44	44	44	44	44	44
ReadFlag			20	20	20	20	20	20
ResetFlag			24	24	24	24	24	24
ReceiveMessage	CCCA	14	112	112	112	194	194	194
	CCCB	15	194	194	194	194	194	194
GetMessageResource			76	76	76	76	76	76
ReleaseMessageResource			76	76	76	76	76	76
GetMessageStatus			78	78	78	78	78	78
SendMessage	SW CCCA	1, 14	130	130	130	226	226	226
	SW CCCB	1, 15	216	216	216	226	226	226
	NS CCCA	14	130	130	130	226	226	226
	NS CCCB	15	216	216	216	226	226	226
	KL CCCA	2, 14	110	110	110	210	210	210
	KL CCCB	2, 15	200	200	200	210	210	210

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
main_dispatch	NoHook	12	144	144	176	144	144	176
	Hook	13	176	176	208	176	176	208
sub_dispatch	B1LI	19	10	10	10	10	10	10
	B1LF	20	14	14	14	14	14	14
	B1HI	21	68	68	68	68	68	68
	B1HF	22	72	72	72	72	72	72
	B2LI	23	n/a	50	70	n/a	50	70
	B2LF	24	n/a	54	74	n/a	54	74
	B2HI	25	n/a	96	148	n/a	96	148
	B2HF	26	n/a	100	152	n/a	100	152
	E1HI	27	n/a	n/a	n/a	238	238	290
	E1HF	28	n/a	n/a	n/a	242	242	294
	E2HI	29	n/a	n/a	n/a	n/a	n/a	290
	E2HF	30	n/a	n/a	n/a	n/a	n/a	294
ErrorHook support		16	82	82	82	82	82	82
	ServiceID	17	94	94	94	94	94	94
	Parameters	18	114	114	114	114	114	114
validity_checks		3	24	24	24	24	24	24
Timing_dispatch		4	66	66	66	66	66	66
Timing_termination		4	68	68	68	68	68	68
ActivateTaskset	SW	1	214	286	318	258	318	362
	NS		256	328	356	300	360	400
	KL	2	168	240	276	212	268	312
ChainTaskset	SWL	1, 8	262	346	366	306	366	414
	SWH	1, 9	308	392	420	348	412	464
	NSL	8	318	402	430	370	422	470
	NSH	9	352	436	464	400	456	508
GetTasksetRef			96	96	96	96	96	96
MergeTaskset			184	184	184	184	184	184
AssignTaskset			134	134	134	134	134	134
RemoveTaskset			184	184	184	184	184	184
TestSubTaskset			184	184	184	184	184	184
TestEquivalentTaskset			184	184	184	184	184	184
TickSchedule	SW	1	254	202	202	202	202	202
	NS		296	250	250	250	250	250
	KL	2	222	162	162	162	162	162
AdvanceSchedule	SW	1	256	200	200	200	200	200
	NS		294	248	248	248	248	248
	KL	2	216	156	156	156	156	156
StartSchedule			162	162	162	162	162	162

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
StopSchedule			118	118	118	118	118	118
GetScheduleStatus			150	150	150	150	150	150
GetScheduleValue			124	124	124	124	124	124
GetScheduleNext			66	66	66	66	66	66
SetScheduleNext			130	130	130	130	130	130
GetArrivalpointDelay			94	94	94	94	94	94
SetArrivalpointDelay			114	114	114	114	114	114
GetArrivalpointTasksetRef			90	90	90	90	90	90
GetArrivalpointNext			94	94	94	94	94	94
SetArrivalpointNext			126	126	126	126	126	126
TestArrivalpointWritable			106	106	106	106	106	106
GetExecutionTime			118	118	118	118	118	118
GetLargestExecutionTime			72	72	72	72	72	72
ResetLargestExecutionTime			68	68	68	68	68	68
GetStackOffset			26	26	26	26	26	26
Interrupt wrapper			136	136	136	136	136	136
Support routines			48	48	48	48	48	48
Support routines			22	22	22	22	22	22
Support routines			98	98	98	98	98	98
Support routines			18	18	18	18	18	18
Support routines			78	78	78	78	78	78
Support routines			8	8	8	8	8	8
Support routines			20	20	20	20	20	20

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.

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, integer tasks.
20	Linked only for basic, single-activation, lightweight, floating point tasks.
21	Linked only for basic, single-activation, heavyweight, integer tasks.
22	Linked only for basic, single-activation, heavyweight, floating point tasks.
23	Linked only for basic, multiple-activation, lightweight, integer tasks.
24	Linked only for basic, multiple-activation, lightweight, floating point tasks.
25	Linked only for basic, multiple-activation, heavyweight, integer tasks.
26	Linked only for basic, multiple-activation, heavyweight, floating point tasks.
27	Linked only for extended, unique priority, integer tasks.
28	Linked only for extended, unique priority, floating point tasks.
29	Linked only for extended, shared priority, integer tasks.
30	Linked only for extended, shared priority, floating point tasks.
31	Implemented as a macro, so no code is linked.
32	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

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	218	263	296	225	257	300
	NS	206	251	284	213	245	288
	KL	45	85	118	52	77	126
TerminateTask	LExt	0	0	0	0	0	0
	H	284	286	295	284	286	295
ChainTask	SWL	378	422	488	443	470	557
	SWH	437	482	552	502	533	616
	NSL	378	422	488	443	470	557
	NSH	429	472	544	494	523	606
Schedule	SW	219	219	233	219	219	233
GetTaskID		27	27	27	27	27	27
GetTaskState		217	217	217	225	225	225
EnableAllInterrupts		45	45	45	45	45	45
DisableAllInterrupts		131	131	131	131	131	131
ResumeAllInterrupts		57	57	57	57	57	57
SuspendAllInterrupts		143	143	143	143	143	143
ResumeOSInterrupts		57	57	57	57	57	57
SuspendOSInterrupts		143	143	143	143	143	143
GetResource	Task	34	34	36	34	34	36
	Combined	123	123	123	123	123	123
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	205	205	205	205	205	205
	Combined	231	231	231	231	231	231
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	234	234	234
	NS	n/a	n/a	n/a	231	231	231
	KL	n/a	n/a	n/a	65	65	65
ClearEvent		n/a	n/a	n/a	117	117	117
GetEvent		n/a	n/a	n/a	196	196	196
WaitEvent	<default>	n/a	n/a	n/a	700	702	771
	fp	n/a	n/a	n/a	707	709	778
GetAlarmBase		254	254	254	250	250	250
GetAlarm		217	217	217	217	217	217
SetRelAlarm		224	224	224	224	224	224
SetAbsAlarm		231	231	231	231	231	231
CancelAlarm		201	201	201	201	201	201
InitCounter		203	203	203	203	203	203
GetCounterValue		206	206	206	206	206	206

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
osek_tick_alarm	<default>	214	214	214	214	214	214
	KL	32	32	32	32	32	32
osek_incr_counter		12	12	12	12	12	12
GetActiveApplicationMode		11	11	11	11	11	11
StartOS		1030	1030	1030	1030	1030	1030
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	114	114	114	114	114	114
InitCOM		8	8	8	8	8	8
CloseCOM		8	8	8	8	8	8
StartCOM		34	34	34	123	123	123
StopCOM		19	19	19	19	19	19
ReadFlag		n/a	n/a	n/a	17	17	17
ResetFlag		n/a	n/a	n/a	12	12	12
ReceiveMessage		194	194	194	383	383	383
GetMessageResource		n/a	n/a	n/a	93	93	93
ReleaseMessageResource		n/a	n/a	n/a	253	253	253
GetMessageStatus		n/a	n/a	n/a	40	40	40
SendMessage	SW	426	471	504	616	648	691
	NS	411	456	489	601	633	676
	KL	94	134	167	283	308	357
ActivateTaskset	SW	205	549	593	212	552	606
	NS	190	534	578	197	537	591
	KL	18	369	412	25	372	425
	SW2	205	549	593	212	552	606
	NS2	190	534	578	197	537	591
	KL2	18	369	412	25	372	425
ChainTaskset	SWL	365	709	785	423	761	853
	SWH	433	776	856	491	828	920
	NSL	365	709	785	423	761	853
	NSH	426	769	849	484	821	913
GetTasksetRef		19	19	19	19	19	19
MergeTaskset		198	198	198	198	198	198
AssignTaskset		15	15	15	15	15	15
RemoveTaskset		192	192	192	192	192	192
TestSubTaskset		206	206	206	206	206	206
TestEquivalentTaskset		203	203	203	203	203	203
TickSchedule	SW	269	634	677	290	648	698
	NS	251	613	656	269	627	677
	KL	84	451	494	107	465	515
	SW2	269	634	677	290	637	690
	NS2	251	613	656	269	616	669
	KL2	84	451	494	107	454	507

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
AdvanceSchedule	SW	249	607	650	263	621	671
	NS	227	586	629	242	600	650
	KL	63	428	471	84	442	492
	SW2	249	607	650	263	610	663
	NS2	227	586	629	242	589	642
	KL2	63	428	471	84	431	484
StartSchedule		217	217	217	217	217	217
StopSchedule		207	207	207	207	207	207
GetScheduleStatus		218	218	218	218	218	218
GetScheduleValue		210	210	210	210	210	210
GetScheduleNext		22	22	22	22	22	22
SetScheduleNext		22	22	22	22	22	22
GetArrivalpointDelay		19	19	19	19	19	19
SetArrivalpointDelay		19	19	19	19	19	19
GetArrivalpointTasksetRef		16	16	16	16	16	16
GetArrivalpointNext		19	19	19	19	19	19
SetArrivalpointNext		19	19	19	19	19	19
TestArrivalpointWritable		30	30	30	30	30	30
GetExecutionTime		8	8	8	8	8	8
GetLargestExecutionTime		14	14	14	14	14	14
ResetLargestExecutionTime		8	8	8	8	8	8
GetStackOffset		30	30	30	30	30	30

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	218	263	296	225	257	300
	NS	206	251	284	213	245	288
	KL	45	85	118	52	77	126
TerminateTask	LExt	0	0	0	0	0	0
	H	604	606	615	604	606	615
ChainTask	SWL	812	856	922	879	906	993
	SWH	781	826	896	848	879	962
	NSL	812	856	922	879	906	993
	NSH	773	816	888	840	869	952

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Schedule	SW	219	219	233	219	219	233
GetTaskID		27	27	27	27	27	27
GetTaskState		217	217	217	225	225	225
EnableAllInterrupts		45	45	45	45	45	45
DisableAllInterrupts		131	131	131	131	131	131
ResumeAllInterrupts		57	57	57	57	57	57
SuspendAllInterrupts		143	143	143	143	143	143
ResumeOSInterrupts		57	57	57	57	57	57
SuspendOSInterrupts		143	143	143	143	143	143
GetResource	Task	34	34	36	34	34	36
	Combined	123	123	123	123	123	123
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	205	205	205	205	205	205
	Combined	231	231	231	231	231	231
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	234	234	234
	NS	n/a	n/a	n/a	231	231	231
	KL	n/a	n/a	n/a	65	65	65
ClearEvent		n/a	n/a	n/a	117	117	117
GetEvent		n/a	n/a	n/a	196	196	196
WaitEvent	<default>	n/a	n/a	n/a	942	944	1016
	fp	n/a	n/a	n/a	949	951	1023
GetAlarmBase		254	254	254	250	250	250
GetAlarm		217	217	217	217	217	217
SetRelAlarm		224	224	224	224	224	224
SetAbsAlarm		231	231	231	231	231	231
CancelAlarm		201	201	201	201	201	201
InitCounter		203	203	203	203	203	203
GetCounterValue		206	206	206	206	206	206
osek_tick_alarm	<default>	214	214	214	214	214	214
	KL	32	32	32	32	32	32
osek_incr_counter		12	12	12	12	12	12
GetActiveApplicationMode		11	11	11	11	11	11
StartOS		2021	2021	2021	2021	2021	2021
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	114	114	114	114	114	114
InitCOM		8	8	8	8	8	8
CloseCOM		8	8	8	8	8	8
StartCOM		34	34	34	123	123	123
StopCOM		19	19	19	19	19	19

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	17	17	17
ResetFlag		n/a	n/a	n/a	12	12	12
ReceiveMessage		194	194	194	383	383	383
GetMessageResource		n/a	n/a	n/a	93	93	93
ReleaseMessageResource		n/a	n/a	n/a	253	253	253
GetMessageStatus		n/a	n/a	n/a	40	40	40
SendMessage	SW	426	471	504	616	648	691
	NS	411	456	489	601	633	676
	KL	94	134	167	283	308	357
ActivateTaskset	SW	205	549	593	212	552	606
	NS	190	534	578	197	537	591
	KL	18	369	412	25	372	425
	SW2	205	549	593	212	552	606
	NS2	190	534	578	197	537	591
	KL2	18	369	412	25	372	425
ChainTaskset	SWL	799	1143	1219	859	1197	1289
	SWH	777	1120	1200	837	1174	1266
	NSL	799	1143	1219	859	1197	1289
	NSH	770	1113	1193	830	1167	1259
GetTasksetRef		19	19	19	19	19	19
MergeTaskset		198	198	198	198	198	198
AssignTaskset		15	15	15	15	15	15
RemoveTaskset		192	192	192	192	192	192
TestSubTaskset		206	206	206	206	206	206
TestEquivalentTaskset		203	203	203	203	203	203
TickSchedule	SW	269	634	677	290	648	698
	NS	251	613	656	269	627	677
	KL	84	451	494	107	465	515
	SW2	269	634	677	290	637	690
	NS2	251	613	656	269	616	669
	KL2	84	451	494	107	454	507
AdvanceSchedule	SW	249	607	650	263	621	671
	NS	227	586	629	242	600	650
	KL	63	428	471	84	442	492
	SW2	249	607	650	263	610	663
	NS2	227	586	629	242	589	642
	KL2	63	428	471	84	431	484
StartSchedule		217	217	217	217	217	217
StopSchedule		207	207	207	207	207	207
GetScheduleStatus		218	218	218	218	218	218

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetScheduleValue		210	210	210	210	210	210
GetScheduleNext		22	22	22	22	22	22
SetScheduleNext		22	22	22	22	22	22
GetArrivalpointDelay		19	19	19	19	19	19
SetArrivalpointDelay		19	19	19	19	19	19
GetArrivalpointTasksetRef		16	16	16	16	16	16
GetArrivalpointNext		19	19	19	19	19	19
SetArrivalpointNext		19	19	19	19	19	19
TestArrivalpointWritable		30	30	30	30	30	30
GetExecutionTime		241	241	241	241	241	241
GetLargestExecutionTime		23	23	23	23	23	23
ResetLargestExecutionTime		17	17	17	17	17	17
GetStackOffset		30	30	30	30	30	30

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	397	439	472	405	433	478
	NS	425	467	500	433	461	506
	KL	227	269	302	235	263	306
TerminateTask	LExt	738	738	747	738	738	747
	H	695	695	704	695	695	704
ChainTask	SWL	1056	1101	1167	1124	1154	1237
	SWH	1026	1068	1138	1094	1124	1206
	NSL	1090	1135	1201	1158	1188	1271
	NSH	1053	1095	1165	1121	1151	1233
Schedule	SW	0	0	0	0	0	0
GetTaskID		36	36	36	36	36	36
GetTaskState		407	407	407	408	408	408
EnableAllInterrupts		61	61	61	61	61	61
DisableAllInterrupts		144	144	144	144	144	144
ResumeAllInterrupts		77	77	77	77	77	77
SuspendAllInterrupts		152	152	152	152	152	152
ResumeOSInterrupts		77	77	77	77	77	77
SuspendOSInterrupts		152	152	152	152	152	152

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetResource	Task	533	533	384	569	569	420
	Combined	285	285	285	321	321	321
	CLEx	383	383	383	419	419	419
ReleaseResource	Task	378	378	378	414	414	414
	Combined	385	385	385	421	421	421
	CLEx	359	359	359	395	395	395
SetEvent	SW	n/a	n/a	n/a	429	429	429
	NS	n/a	n/a	n/a	443	443	443
	KL	n/a	n/a	n/a	268	268	268
ClearEvent		n/a	n/a	n/a	245	245	245
GetEvent		n/a	n/a	n/a	407	407	407
WaitEvent	<default>	n/a	n/a	n/a	1032	1032	1074
	fp	n/a	n/a	n/a	1039	1039	1081
GetAlarmBase		385	385	385	381	381	381
GetAlarm		344	344	344	344	344	344
SetRelAlarm		373	373	373	373	373	373
SetAbsAlarm		375	375	375	375	375	375
CancelAlarm		325	325	325	325	325	325
InitCounter		327	327	327	327	327	327
GetCounterValue		325	325	325	325	325	325
osek_tick_alarm	<default>	233	233	233	233	233	233
	KL	32	32	32	32	32	32
osek_incr_counter		12	12	12	12	12	12
GetActiveApplicationMode		11	11	11	11	11	11
StartOS		2121	2121	2121	2121	2121	2121
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	120	120	120	120	120	120
InitCOM		8	8	8	8	8	8
CloseCOM		8	8	8	8	8	8
StartCOM		54	54	54	140	140	140
StopCOM		34	34	34	34	34	34
ReadFlag		n/a	n/a	n/a	30	30	30
ResetFlag		n/a	n/a	n/a	26	26	26
ReceiveMessage		295	295	295	484	484	484
GetMessageResource		n/a	n/a	n/a	542	542	542
ReleaseMessageResource		n/a	n/a	n/a	538	538	538
GetMessageStatus		n/a	n/a	n/a	123	123	123
SendMessage	SW	704	746	779	895	923	968
	NS	729	771	804	920	948	993
	KL	377	419	452	568	596	639

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ActivateTaskset	SW	615	993	1023	645	988	1048
	NS	634	1015	1040	666	1010	1069
	KL	442	818	853	469	817	873
	SW2	615	993	1023	645	988	1048
	NS2	634	1015	1040	666	1010	1069
	KL2	442	818	853	469	817	873
ChainTaskset	SWL	1292	1674	1741	1479	1724	1817
	SWH	1269	1650	1716	1353	1701	1798
	NSL	1329	1710	1775	1515	1762	1854
	NSH	1297	1679	1747	1481	1729	1826
GetTasksetRef		197	197	197	197	197	197
MergeTaskset		281	281	281	281	281	281
AssignTaskset		92	92	92	92	92	92
RemoveTaskset		275	275	275	275	275	275
TestSubTaskset		285	285	285	285	285	285
TestEquivalentTaskset		282	282	282	282	282	282
TickSchedule	SW	351	1124	1159	775	1145	1198
	NS	372	1146	1181	797	1167	1220
	KL	176	951	986	602	972	1025
	SW2	351	1124	1159	775	1123	1179
	NS2	372	1146	1181	797	1145	1201
	KL2	176	951	986	602	950	1006
AdvanceSchedule	SW	332	1106	1141	757	1127	1180
	NS	351	1128	1163	779	1149	1202
	KL	162	936	971	587	957	1010
	SW2	332	1106	1141	757	1105	1161
	NS2	351	1128	1163	779	1127	1183
	KL2	162	936	971	587	935	991
StartSchedule		0	0	0	0	0	0
StopSchedule		0	0	0	0	0	0
GetScheduleStatus		0	0	0	0	0	0
GetScheduleValue		0	0	0	0	0	0
GetScheduleNext		0	0	0	0	0	0
SetScheduleNext		0	0	0	0	0	0
GetArrivalpointDelay		0	0	0	0	0	0
SetArrivalpointDelay		0	0	0	0	0	0
GetArrivalpointTasksetRef		0	0	0	0	0	0
GetArrivalpointNext		0	0	0	0	0	0
SetArrivalpointNext		0	0	0	0	0	0
TestArrivalpointWritable		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	
GetExecutionTime		269	269	269	269	269	269
GetLargestExecutionTime		186	186	186	186	186	186
ResetLargestExecutionTime		178	178	178	178	178	178
GetStackOffset		30	30	30	30	30	30

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	18	18	18	18	18	18
	Cat 2	118	118	118	118	118	118

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	18	18	18	18	18	18
	Cat 2	225	225	225	225	225	225

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	18	18	18	18	18	18
	Cat 2	225	225	225	225	225	225

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.

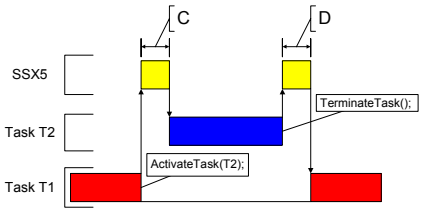


Figure 1: Task Activates a Higher Priority Task which Terminates Normally

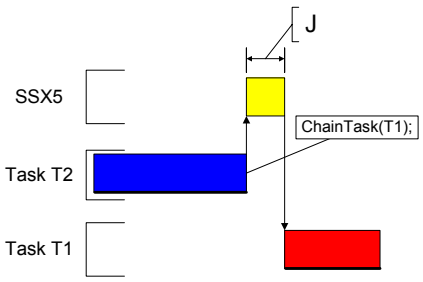


Figure 2: Task Chaining

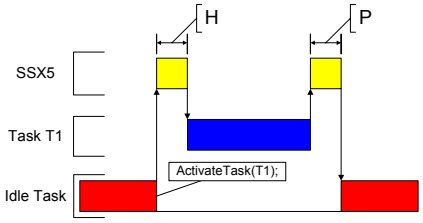


Figure 3: Task Activation from Idle Task

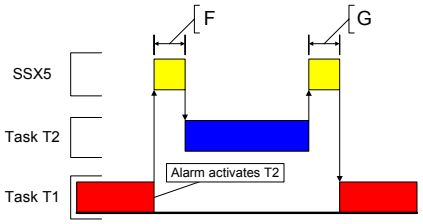


Figure 4: Task Activation from an Alarm

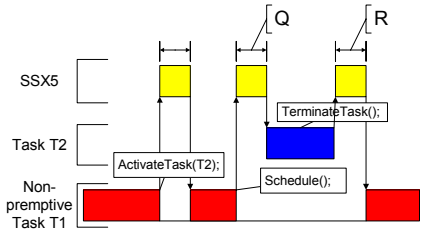


Figure 5: Non-Preemptive Task Calls Schedule()

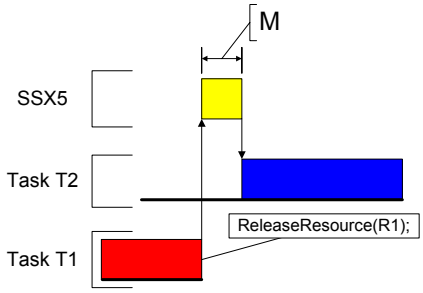


Figure 6: Blocked Task Activated by ReleaseResource()

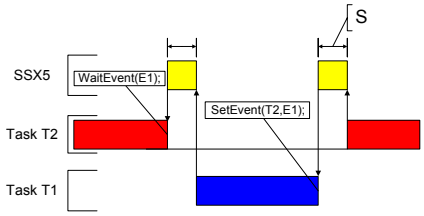


Figure 7: Waiting Task Activated by SetEvent()

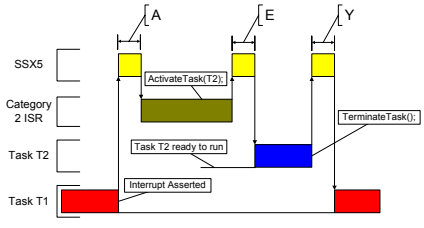


Figure 8: Category 2 ISR Activates a Higher Priority Task

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	211	315	345	211	315	345
Figure 1: D	Heavy, Basic/Extended	287	387	412	393	395	419
ChainTask	Light, Basic	313	374	442	320	374	458
Figure 2: J	Heavy, Basic/Extended	687	849	946	800	857	964
Pre-emption	Light, Basic	216	278	353	223	277	362
Figure 1: C	Heavy, Basic/Extended	285	330	403	350	380	465
From idle task	Light, Basic	219	281	356	226	280	365
Figure 3: H	Heavy, Basic/Extended	288	333	406	353	383	468
Triggered by alarm	Light, Basic	445	507	582	452	506	591
Figure 4: F	Heavy, Basic/Extended	514	559	632	579	609	694
Schedule	Light, Basic	209	228	284	209	228	284
Figure 5: Q	Heavy, Basic/Extended	278	280	334	336	336	392
Release resource	Light, Basic	217	236	278	217	236	278
Figure 6: M	Heavy, Basic/Extended	286	288	328	344	344	386
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	578	578	709
From category 2 ISR	Light, Basic	112	131	173	112	131	173
Figure 8: E	Heavy, Basic/Extended	181	183	223	239	239	281

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	621	644	674	621	644	674
Figure 1: D	Heavy, Basic/Extended	607	629	654	635	637	664
ChainTask	Light, Basic	763	805	873	770	805	889
Figure 2: J	Heavy, Basic/Extended	1367	1432	1529	1402	1440	1550
Pre-emption	Light, Basic	347	390	465	354	389	474
Figure 1: C	Heavy, Basic/Extended	400	445	518	467	497	582
From idle task	Light, Basic	350	393	468	357	392	477
Figure 3: H	Heavy, Basic/Extended	403	448	521	470	500	585

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations	Task Attributes	No	Yes		No	Yes	
Triggered by alarm	Light, Basic	577	620	695	584	619	704
Figure 4: F	Heavy, Basic/Extended	630	675	748	697	727	812
Schedule	Light, Basic	340	340	396	340	340	396
Figure 5: Q	Heavy, Basic/Extended	393	395	449	453	453	509
Release resource	Light, Basic	348	348	390	348	348	390
Figure 6: M	Heavy, Basic/Extended	401	403	443	461	461	503
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	669	669	800
From category 2 ISR	Light, Basic	651	651	693	651	651	693
Figure 8: E	Heavy, Basic/Extended	704	706	746	764	764	806

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	734	755	785	734	755	785
Figure 1: D	Heavy, Basic/Extended	698	718	743	726	726	753
ChainTask	Light, Basic	1007	1050	1118	1015	1050	1133
Figure 2: J	Heavy, Basic/Extended	1703	1763	1860	1739	1771	1880
Pre-emption	Light, Basic	518	560	635	526	559	643
Figure 1: C	Heavy, Basic/Extended	571	615	688	639	667	754
From idle task	Light, Basic	521	563	638	529	562	646
Figure 3: H	Heavy, Basic/Extended	574	618	691	642	670	757
Triggered by alarm	Light, Basic	767	809	884	775	808	892
Figure 4: F	Heavy, Basic/Extended	820	864	937	888	916	1003
Schedule	Light, Basic	387	387	443	387	387	443
Figure 5: Q	Heavy, Basic/Extended	440	442	496	500	500	556

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	509	509	551	545	545	587
Figure 6: M	Heavy, Basic/Extended	562	564	604	658	658	700
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	875	875	982
From category 2 ISR	Light, Basic	679	679	721	679	679	721
Figure 8: E	Heavy, Basic/Extended	732	734	774	792	792	834

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	72	72	80	72	72	80
BCC1 lightweight, floating point	80	80	88	80	80	88
BCC1 heavyweight, integer	208	208	216	208	208	216
BCC1 heavyweight, floating point	208	208	216	208	208	216
BCC2 lightweight, integer	n/a	80	88	n/a	80	88
BCC2 lightweight, floating point	n/a	80	88	n/a	80	88
BCC2 heavyweight, integer	n/a	216	216	n/a	216	216
BCC2 heavyweight, floating point	n/a	216	216	n/a	216	216
ECC1 heavyweight, integer	n/a	n/a	n/a	248	248	256
ECC1 heavyweight, floating point	n/a	n/a	n/a	248	248	256
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	256
ECC2 heavyweight, floating point	n/a	n/a	n/a	n/a	n/a	256

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		80	80	80	80	80	80
BCC1 lightweight, floating point		88	88	88	88	88	88
BCC1 heavyweight, integer		216	216	216	216	216	216
BCC1 heavyweight, floating point		216	216	216	216	216	216
BCC2 lightweight, integer		n/a	88	88	n/a	88	88
BCC2 lightweight, floating point		n/a	88	88	n/a	88	88
BCC2 heavyweight, integer		n/a	224	216	n/a	224	216
BCC2 heavyweight, floating point		n/a	224	216	n/a	224	216
ECC1 heavyweight, integer		n/a	n/a	n/a	256	256	256
ECC1 heavyweight, floating point		n/a	n/a	n/a	256	256	256
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	256
ECC2 heavyweight, floating point		n/a	n/a	n/a	n/a	n/a	256

Timing

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		104	104	112	104	104	112
BCC1 lightweight, floating point		104	104	112	104	104	112
BCC1 heavyweight, integer		232	232	240	232	232	240
BCC1 heavyweight, floating point		232	232	240	232	232	240
BCC2 lightweight, integer		n/a	104	112	n/a	104	112
BCC2 lightweight, floating point		n/a	104	112	n/a	104	112
BCC2 heavyweight, integer		n/a	240	240	n/a	240	240
BCC2 heavyweight, floating point		n/a	240	240	n/a	240	240
ECC1 heavyweight, integer		n/a	n/a	n/a	272	272	280
ECC1 heavyweight, floating point		n/a	n/a	n/a	272	272	280
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	280
ECC2 heavyweight, floating point		n/a	n/a	n/a	n/a	n/a	280
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		112	112	112	112	112	112
BCC1 lightweight, floating point		112	112	112	112	112	112

Configuration / Events / Shared Task Priorities / Multiple Task Activations		Application Uses					
		No			Yes		
		No		Yes	No		Yes
		No	Yes		No	Yes	
BCC1 heavyweight, integer		240	240	240	240	240	240
BCC1 heavyweight, floating point		240	240	240	240	240	240
BCC2 lightweight, integer		n/a	112	112	n/a	112	112
BCC2 lightweight, floating point		n/a	112	112	n/a	112	112
BCC2 heavyweight, integer		n/a	248	240	n/a	248	240
BCC2 heavyweight, floating point		n/a	248	240	n/a	248	240
ECC1 heavyweight, integer		n/a	n/a	n/a	280	280	280
ECC1 heavyweight, floating point		n/a	n/a	n/a	280	280	280
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	280
ECC2 heavyweight, floating point		n/a	n/a	n/a	n/a	n/a	280

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		104	104	112	104	104	112
BCC1 lightweight, floating point		104	104	112	104	104	112
BCC1 heavyweight, integer		232	232	240	232	232	240
BCC1 heavyweight, floating point		232	232	240	232	232	240
BCC2 lightweight, integer		n/a	104	112	n/a	104	112
BCC2 lightweight, floating point		n/a	104	112	n/a	104	112
BCC2 heavyweight, integer		n/a	240	240	n/a	240	240
BCC2 heavyweight, floating point		n/a	240	240	n/a	240	240
ECC1 heavyweight, integer		n/a	n/a	n/a	272	272	280
ECC1 heavyweight, floating point		n/a	n/a	n/a	272	272	280
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	280
ECC2 heavyweight, floating point		n/a	n/a	n/a	n/a	n/a	280
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		112	112	112	112	112	112
BCC1 lightweight, floating point		112	112	112	112	112	112
BCC1 heavyweight, integer		240	240	240	240	240	240
BCC1 heavyweight, floating point		240	240	240	240	240	240
BCC2 lightweight, integer		n/a	112	112	n/a	112	112

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC2 lightweight, floating point		n/a	112	112	n/a	112	112
BCC2 heavyweight, integer		n/a	248	240	n/a	248	240
BCC2 heavyweight, floating point		n/a	248	240	n/a	248	240
ECC1 heavyweight, integer		n/a	n/a	n/a	280	280	280
ECC1 heavyweight, floating point		n/a	n/a	n/a	280	280	280
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	280
ECC2 heavyweight, floating point		n/a	n/a	n/a	n/a	n/a	280

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.

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