LiveDevices

Binding Manual: TMS470/TI

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

USA

ETAS Inc.
3021 Miller Road
Ann Arbor, MI 48103

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

www.etasinc.com

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

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

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

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

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

Contents

1 About this Guide

This guide provides port specific information for the TMS470/TI implementation of Realogy Real-Time Architect (RTA).

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 the low-level technical information to integrate SSX5 into your application.

1.2 Conventions

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 `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	Texas Instruments
Compiler	TMS470R1x C/C++ Compiler
Version	2.30

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**
`-mt`	Compile in 16-bit assembly instructions.

2.1.1 Using the 16-Bit/32-Bit Assembler Instruction Set

The TI compiler can generate object code using either the TMS470 16-bit or 32-bit instruction set. The SSX5 run-time libraries have been compiled using the 16-bit instruction set to reduce the amount of code memory used. The libraries support applications using either the 16-bit or the 32-bit instruction set.

All C files that contain tasks (i.e. functions decorated with the OSEK standard `TASK()` macro), Category 2 ISRs (i.e. functions decorated with the OSEK standard `ISR()` macro), function callbacks and hooks must be compiled using either the 16-bit instruction set or the 32-bit instruction set using inter-working support.

2.2 Assembler

SSX5 was built using the following assembler:

Vendor	Texas Instruments
Assembler	TMS470 COFF Assembler
Version	2.30

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

2.3 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_intvec`	ROM	Vector table "if generated by RTArchitect".
`os_pir`	RAM	SSX5 initialized data.
`os_pur`	RAM	SSX5 uninitialized data.

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

C Library Functions	Description
`IND$CALL`	Indirect function call.
`IND_CALL`	

2.3.1 Stack Initialization for Different CPU Modes

The TMS470 CPU can use up to six different stacks, depending upon the number of operating modes that occur in an application.

RTA uses two modes, supervisor (SVC) mode and IRQ mode (when Category 2 interrupts occur in an application). FIQ mode will only be used in applications that use Category 1 FIQ interrupts and, therefore, operate outside of the scope of RTA.

All stack pointers must be initialized before use. It should be noted that it is your responsibility to ensure that each stack is large enough to avoid overrun.

The example application demonstrates a suggested stack pointer initialization method. The start-up code, `init.asm`, declares the each stack section with the lengths defined by `STACK_LEN_<mode>` values. The linker command file, `linktms.cmd`, locates the stack's sections in memory and defines labels at the top of each stack section, `_os_stack_top_<mode>`. The `_os_stack_top_svc` and `_os_stack_top_irq` labels must be defined in all applications that use ECC tasks.

2.3.2 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 `NOLOAD` in the linker command file. This prevents the contents, relocation information and line number information being placed in the output module.

The linker allocates space for the section and its symbols and it appears in the memory map listing. The example application contains an example of a linker command file.

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 *TMS470R1x User's Guide*.

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

IPL Values	CPSR I-Bit	CPSR F-Bit	Description
0	0	0	User Level.
N/A	0	1	Not Allowed.
1	1	0	OS Level - IRQ Category 1 and 2 interrupt level.
2	1	1	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	Legality
0x04	Undefined instructions	Category 2.
0x08	SWI	Category 2.
0x0C	Prefetch Abort	Category 2.
0x10	Data Abort	Category 2.
0x18	IRQ (General Interrupt)	Single Category 1 or 2.
0x1C	FIQ (Fast Interrupt)	Single Category 1.
0x20	CIM channel 0	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x24	CIM channel 1	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x28	CIM channel 2	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x2C	CIM channel 3	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x30	CIM channel 4	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x34	CIM channel 5	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x38	CIM channel 6	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x3C	CIM channel 7	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x40	CIM channel 8	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x44	CIM channel 9	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x48	CIM channel 10	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x4C	CIM channel 11	Category 1 or 2 (Category 2 only for IRQ interrupts).

Vector	Description	Legality
0x50	CIM channel 12	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x54	CIM channel 13	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x58	CIM channel 14	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x5C	CIM channel 15	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x60	CIM channel 16	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x64	CIM channel 17	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x68	CIM channel 18	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x6C	CIM channel 19	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x70	CIM channel 20	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x74	CIM channel 21	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x78	CIM channel 22	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x7C	CIM channel 23	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x80	CIM channel 24	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x84	CIM channel 25	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x88	CIM channel 26	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x8C	CIM channel 27	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x90	CIM channel 28	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x94	CIM channel 29	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x98	CIM channel 30	Category 1 or 2 (Category 2 only for IRQ interrupts).
0x9C	CIM channel 31	Category 1 or 2 (Category 2 only for IRQ interrupts).

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 TI 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 standard `ISR()` macro, shown in Code Example 3:1.

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

Code Example 3:1 - Category 2 C Function Context

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.asm`. 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 (`VV` represents the 2 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
`0xVV`	`_os_wrapper_VV`
`e.g. 0x24`	`_os_wrapper_24`

3.1.6 Software Vector Table

The vector table used in the TMS470R1 implementation of RTA concatenates the ARM CPU vector table with the 32 interrupt channels of the Central Interrupt Manager (CIM) (see Section 3.1.2).

The 32 CIM channels are characterized in an integer array `os_irq_fiq_vectors`, which contains the addresses of the interrupt handlers for ISRs on these vectors. The offset addresses, shown in Section 3.1.2, illustrate the direct mapping between each array entry and the interrupt source that should be used in the OIL configuration file.

3.1.7 Reset Vector

In the generated vector table, the Reset Vector at 0x0 is always attached to a user function `c_int00()`. This reset handler function should perform the start-up operation required for the TI C compiler and setup the stack pointers for the CPU modes used. An example of `c_int00()` is provided in the example application.

3.1.8 Handling Multiple Interrupt Sources

The CIM allows multiple interrupt sources to be processed and handled by the CPU FIQ and IRQ interrupt lines. RTA supports this using the `os_irq_entry()` and `os_fiq_entry()` functions. These functions are the multiple interrupt handler entry points. They reference the CIM hardware registers to determine the interrupt source.

If RTA generates the vector table and multiple FIQ and IRQ interrupts are contained within the application OIL configuration file, these functions are automatically located on vector table addresses 0x18 and 0x1C. If a user generated vector table is provided for an application with multiple interrupts bound on FIQ/IRQ, the appropriate functions should be bound to addresses 0x18 and 0x1C.

When multiple interrupts occur within the application, the entry functions of all Category 1 interrupts should be standard C functions. They should not be decorated with the `interrupt` modifier. Category 2 interrupts should still be decorated with the OSEK standard `ISR()` macro. Extra context handling required by interrupt functions is dealt with by `os_irq_entry()` and `os_fiq_entry()`.

3.1.9 Handling Single IRQ/FIQ Interrupt Sources

The CIM does not need to be referenced to determine the interrupt source in applications where single interrupts trigger the IRQ or FIQ interrupt line. Instead, the interrupt can be bound directly to the CPU IRQ (0x18) or FIQ (0x1C) vector, consequently reducing the interrupt entry time. Single Category 2 IRQ interrupts should be bound to vector 0x18 in the OIL configuration file.

The entry function that occurs on the wrapper is `_os_wrapper_018()`. If a user generated vector table is used with a single Category 2 IRQ interrupt, the assembler in Code Example 3:2 can be used to implement this vector table.

```
     .sect ".os_intvec"
     .align     4
     LDR  pc,_os_reset_addr
     B    $
     B    $
     B    $
     B    $
     NOP
     LDR  pc,_os_irq_addr
     B    $
     .ref  _c_int00
  _os_reset_addr:    .long _c_int00
  _os_irq_addr:      .long _os_wrapper_018
```

Code Example 3:2 - Implementing a Vector Table with a Single Category 2 ISR

3.2 Register Settings

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

SSX5 does not reserve the use of any hardware registers.

3.3 CPU Operating Modes

All tasks and Category 2 ISRs execute in supervisor mode. When a Category 2 interrupt occurs, SSX5 switches from IRQ mode to supervisor (SVC) mode before processing the interrupt. This minimizes the worst-case stack requirements.

From reset, the processor runs on the SVC stack. SSX5 always expects to run on the SVC stack (i.e. in SVC mode), except when processing Category 1 interrupts.

Category 1 interrupts can be configured to use the following CPU operating modes; SVC by using the SWI, FIQ, IRQ, Abort and Undefined.

3.4 Stack Usage

3.4.1 Number of Stacks

Two stacks are used. The SVC stack is indexed 0 and the IRQ stack is indexed 1.

The first argument to `StackFaultHook` is 0 or 1. This indicates the stack on which the excess was observed.

`StackOffsetType` is a structure of two scalars, representing the number of bytes on each stack. This is shown in Code Example 3:3.

```
typedef struct {
    osStackBytes0 svc;
    osStackBytes1 irq;
} StackOffsetType;
```

Code Example 3:3 - 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

Stack	API Max Usage (Bytes)
SVC_stack	68
IRQ_stack	0

Timing

Stack	API Max Usage (Bytes)
SVC_stack	72
IRQ_stack	0

Extended

Stack	API Max Usage (Bytes)
SVC_stack	76
IRQ_stack	0

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.

3.4.3 Stack Usage with 16-Bit/32-Bit Instructions

The SSX5 libraries can support applications written in both the 16-bit and 32-bit TMS470 instruction sets. The libraries were compiled using the 16-bit instruction set. Stack usage will change when applications use 32-bit instructions.

The 16-bit instruction set should be used to optimize stack usage. When the 32-bit instruction set is used, veneers are placed between functions compiled using the 16 and 32-bit instruction sets. Each veneer uses 4 bytes of stack, which should be taken into account during SVC mode stack calculations.

When an ECC task occurs in an application the amount of stack used by the task must be entered into the OIL configuration file. If the task is compiled using 32-bit instructions, the stack figure in the OIL file is the amount of stack used by the task with 8 bytes added (4 bytes for the veneer to get into the task and 4 bytes for the veneer to enter the `WaitEvent()` API call).

3.4.4 Stack Usage in Tick_<Counter Name> ()

In applications that use alarms, a function `Tick_<counter name>` (where `<counter name>` is the associated counter that the alarm is attached to) is placed in `osekdefs.c`.

The amount of stack used in this function depends upon the level of compiler optimization and the number of alarms implemented. A nominal stack usage of 16 bytes has been attributed to this function, which is sufficient for up to 8 alarms. If more alarms are used within an application that uses ECC tasks, then the number of stack bytes used by `Tick_<counter name>()` should be reviewed. Any extra stack usage should be added to the idle task stack figure in the OIL configuration file.

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.

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

Processor	**TMS470**
Clock speed (MHz)	7
Code memory	Off-chip RAM
Read-only data memory	Off-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)	4294967295					

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		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
OS overhead	RAM	28	28	28	28	28	28
	ROM	154	154	154	154	154	154
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Timing

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
OS overhead	RAM	48	48	48	48	48	48
	ROM	226	226	226	226	226	226
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

Extended

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
OS overhead	RAM	66	66	66	66	66	66
	ROM	272	272	272	272	272	272
COM overhead	RAM	8	8	8	8	8	8
	ROM	16	16	16	16	16	16

4.2.2 ROM and RAM for OSEK OS Objects

In addition to the base OS overhead, detailed in Section 4.2.1, each OSEK OS object requires ROM and/or RAM. 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	56	56	56
	ROM	n/a	n/a	n/a	72	72	72
ECC1, floating point task	RAM	n/a	n/a	n/a	58	58	58
	ROM	n/a	n/a	n/a	72	72	72
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	58
	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	60
	ROM	n/a	n/a	n/a	n/a	n/a	80
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	48	48	48	48	48	48
Category 2 ISR, floating point	RAM	1	1	1	1	1	1
	ROM	64	64	64	64	64	64

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	12	12	12	12	12	12
	ROM	36	36	36	36	36	36
Counter	RAM	4	4	4	4	4	4
	ROM	40	40	40	40	40	40
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	12	12	12	12	12	12
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	12	12	12	12	12	12
	ROM	48	48	48	48	48	48
BCC1 Heavyweight task	RAM	16	16	16	16	16	16
	ROM	52	52	52	52	52	52
BCC2 task	RAM	n/a	20	22	n/a	20	22
	ROM	n/a	56	64	n/a	56	64
ECC1, Integer task	RAM	n/a	n/a	n/a	68	68	68
	ROM	n/a	n/a	n/a	84	84	84
ECC1, floating point task	RAM	n/a	n/a	n/a	70	70	70
	ROM	n/a	n/a	n/a	84	84	84
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	70
	ROM	n/a	n/a	n/a	n/a	n/a	92
ECC2, floating point task	RAM	n/a	n/a	n/a	n/a	n/a	72
	ROM	n/a	n/a	n/a	n/a	n/a	92
Category 2 ISR	RAM	12	12	12	12	12	12
	ROM	82	82	82	82	82	82
Category 2 ISR, floating point	RAM	14	14	14	14	14	14
	ROM	92	92	92	92	92	92
Resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Alarm	RAM	12	12	12	12	12	12
	ROM	36	36	36	36	36	36
Counter	RAM	4	4	4	4	4	4
	ROM	40	40	40	40	40	40
Message	RAM	11	11	11	31	31	31
	ROM	20	20	20	56	56	56
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	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**	
Arrivalpoint (writable)	RAM	12	12	12	12	12	12
	ROM	12	12	12	12	12	12
Schedule	RAM	16	16	16	16	16	16
	ROM	36	36	36	36	36	36
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

Extended

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
BCC1 Lightweight task	RAM	16	16	16	16	16	16
	ROM	60	60	60	60	60	60
BCC1 Heavyweight task	RAM	20	20	20	20	20	20
	ROM	60	60	60	60	60	60
BCC2 task	RAM	n/a	24	26	n/a	24	26
	ROM	n/a	64	72	n/a	64	72
ECC1, Integer task	RAM	n/a	n/a	n/a	72	72	72
	ROM	n/a	n/a	n/a	92	92	92
ECC1, floating point task	RAM	n/a	n/a	n/a	74	74	74
	ROM	n/a	n/a	n/a	92	92	92
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	74
	ROM	n/a	n/a	n/a	n/a	n/a	100
ECC2, floating point task	RAM	n/a	n/a	n/a	n/a	n/a	76
	ROM	n/a	n/a	n/a	n/a	n/a	100
Category 2 ISR	RAM	16	16	16	16	16	16
	ROM	94	94	94	94	94	94
Category 2 ISR, floating point	RAM	18	18	18	18	18	18
	ROM	104	104	104	104	104	104
Resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Alarm	RAM	12	12	12	12	12	12
	ROM	40	40	40	40	40	40

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Counter	RAM	4	4	4	4	4	4
	ROM	44	44	44	44	44	44
Message	RAM	11	11	11	31	31	31
	ROM	24	24	24	60	60	60
Flag	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4
Message resource	RAM	8	8	8	8	8	8
	ROM	28	28	28	28	28	28
Event	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Priority level	RAM	0	0	6	0	6	6
	ROM	0	0	12	0	12	12
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	20	20	20	20	20	20
Arrivalpoint (writable)	RAM	20	20	20	20	20	20
	ROM	20	20	20	20	20	20
Schedule	RAM	20	20	20	20	20	20
	ROM	44	44	44	44	44	44
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	4	4	4	4	4	4
Taskset (writable)	RAM	4	4	4	4	4	4
	ROM	4	4	4	4	4	4

4.2.3 Size of Linkable Modules

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.

Variant	Description
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.
No Params	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
No ServiceID	`ErrorHook` does not use `GetServiceID` or `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	120	156	190	130	166	216
	NS		100	136	170	110	146	196
	KL	2	72	108	142	82	118	168
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	46	46	46	46	46	46
ChainTask	SWL	1, 8	102	146	176	112	156	202
	SWH	1, 9	130	166	196	140	176	226
	NSL	8	102	146	176	112	156	202
	NSH	9	118	154	184	128	164	214
Schedule			96	96	116	96	96	116
GetTaskID			40	40	40	40	40	40
GetTaskState			92	92	92	108	108	108
EnableAllInterrupts			40	40	40	40	40	40
DisableAllInterrupts			48	48	48	48	48	48

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
ResumeAllInterrupts			54	54	54	54	54	54
SuspendAllInterrupts			66	66	66	66	66	66
ResumeOSInterrupts			54	54	54	54	54	54
SuspendOSInterrupts			66	66	66	66	66	66
GetResource	Task	7	36	36	40	36	36	40
	Combined	6	76	76	76	76	76	76
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	80	80	80	80	80	80
	Combined	6	80	80	80	80	80	80
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	122	122	194
	NS		n/a	n/a	n/a	102	102	174
	NS1i	10	n/a	n/a	n/a	66	n/a	n/a
	KL	2	n/a	n/a	n/a	80	80	152
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	58	58	58
GetEvent			n/a	n/a	n/a	58	58	58
WaitEvent	<default>		n/a	n/a	n/a	226	226	382
	fp	11	n/a	n/a	n/a	248	248	438
	1i	10	n/a	n/a	n/a	36	n/a	n/a
GetAlarmBase			60	60	60	60	60	60
GetAlarm			94	94	94	94	94	94
SetRelAlarm			100	100	100	100	100	100
SetAbsAlarm			112	112	112	112	112	112
CancelAlarm			82	82	82	82	82	82
InitCounter			70	70	70	70	70	70
GetCounterValue			82	82	82	82	82	82
osek_tick_alarm	<default>		80	80	80	80	80	80
	KL	2	46	46	46	46	46	46
osek_incr_counter			40	40	40	40	40	40
GetActiveApplicationMode		31	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			136	136	136	136	136	136
ShutdownOS	NoHook	12	50	50	50	50	50	50
	Hook	13	58	58	58	58	58	58
InitCOM			16	16	16	16	16	16
CloseCOM			16	16	16	16	16	16
StartCOM			42	42	42	42	42	42
StopCOM			32	32	32	32	32	32
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	68	68	68	160	160	160
	CCCB	15	160	160	160	160	160	160

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
GetMessageResource			62	62	62	62	62	62
ReleaseMessageResource			58	58	58	58	58	58
GetMessageStatus			50	50	50	50	50	50
SendMessage	SW CCCA	1, 14	88	88	88	190	190	190
	SW CCCB	1, 15	178	178	178	190	190	190
	NS CCCA	14	88	88	88	190	190	190
	NS CCCB	15	178	178	178	190	190	190
	KL CCCA	2, 14	66	66	66	166	166	166
	KL CCCB	2, 15	154	154	154	166	166	166
main_dispatch	NoHook	12	118	118	148	118	118	148
	Hook	13	154	154	184	154	154	184
sub_dispatch	B1LI	19	42	42	42	42	42	42
	B1LF	20	50	50	50	50	50	50
	B1HI	21	94	94	94	94	94	94
	B1HF	22	102	102	102	102	102	102
	B2LI	23	n/a	76	100	n/a	76	100
	B2LF	24	n/a	84	108	n/a	84	108
	B2HI	25	n/a	134	182	n/a	134	182
	B2HF	26	n/a	142	190	n/a	142	190
	E1HI	27	n/a	n/a	n/a	378	378	426
	E1HF	28	n/a	n/a	n/a	386	386	434
	E2HI	29	n/a	n/a	n/a	n/a	n/a	426
	E2HF	30	n/a	n/a	n/a	n/a	n/a	434
CAT2_wrapper			58	58	58	58	58	58
hook_support	No ServiceID	16	56	56	56	56	56	56
	No Parameters	17	64	64	64	64	64	64
		18	76	76	76	76	76	76
fp_support			80	80	80	80	80	80
utility_functions	common		96	96	96	96	96	96
	optional	32	n/a	n/a	n/a	n/a	n/a	n/a
	optional	32	74	74	74	74	74	74
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	84	128	164	94	148	188
	NS		64	108	142	74	128	166
	KL	2	30	88	122	40	108	146
ChainTaskset	SWL	1, 8	66	110	144	66	128	158
	SWH	1, 9	100	146	182	100	156	196
	NSL	8	66	110	144	66	128	158
	NSH	9	88	134	170	88	144	184

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetTasksetRef			20	20	20	20	20	20
MergeTaskset			60	60	60	60	60	60
AssignTaskset			20	20	20	20	20	20
RemoveTaskset			60	60	60	60	60	60
TestSubTaskset			70	70	70	70	70	70
TestEquivalentTaskset			68	68	68	68	68	68
TickSchedule	SW	1	132	138	138	138	138	138
	NS		110	110	110	110	110	110
	KL	2	90	92	92	92	92	92
AdvanceSchedule	SW	1	126	126	126	126	126	126
	NS		104	98	98	98	98	98
	KL	2	82	80	80	80	80	80
StartSchedule			84	84	84	84	84	84
StopSchedule			68	68	68	68	68	68
GetScheduleStatus			96	96	96	96	96	96
GetScheduleValue			74	74	74	74	74	74
GetScheduleNext			22	22	22	22	22	22
SetScheduleNext			20	20	20	20	20	20
GetArrivalpointDelay			20	20	20	20	20	20
SetArrivalpointDelay			18	18	18	18	18	18
GetArrivalpointTasksetRef			18	18	18	18	18	18
GetArrivalpointNext			20	20	20	20	20	20
SetArrivalpointNext			18	18	18	18	18	18
TestArrivalpointWritable			42	42	42	42	42	42
GetExecutionTime			16	16	16	16	16	16
GetLargestExecutionTime			18	18	18	18	18	18
ResetLargestExecutionTime			16	16	16	16	16	16
GetStackOffset			40	40	40	40	40	40

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	120	156	190	130	166	216
	NS		100	136	170	110	146	196
	KL	2	72	108	142	82	118	168
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	46	46	46	46	46	46

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
ChainTask	SWL	1, 8	102	146	176	112	156	202
	SWH	1, 9	130	166	196	140	176	226
	NSL	8	102	146	176	112	156	202
	NSH	9	118	154	184	128	164	214
Schedule			116	116	136	116	116	136
GetTaskID			40	40	40	40	40	40
GetTaskState			92	92	92	108	108	108
EnableAllInterrupts			40	40	40	40	40	40
DisableAllInterrupts			48	48	48	48	48	48
ResumeAllInterrupts			54	54	54	54	54	54
SuspendAllInterrupts			66	66	66	66	66	66
ResumeOSInterrupts			54	54	54	54	54	54
SuspendOSInterrupts			66	66	66	66	66	66
GetResource	Task	7	36	36	40	36	36	40
	Combined	6	76	76	76	76	76	76
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	100	100	100	100	100	100
	Combined	6	100	100	100	100	100	100
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	122	122	194
	NS		n/a	n/a	n/a	102	102	174
	NS1i	10	n/a	n/a	n/a	66	n/a	n/a
	KL	2	n/a	n/a	n/a	80	80	152
	KL1i	2, 10	n/a	n/a	n/a	32	n/a	n/a
ClearEvent			n/a	n/a	n/a	58	58	58
GetEvent			n/a	n/a	n/a	58	58	58
WaitEvent	<default>		n/a	n/a	n/a	226	226	382
	fp	11	n/a	n/a	n/a	248	248	438
	1i	10	n/a	n/a	n/a	36	n/a	n/a
GetAlarmBase			60	60	60	60	60	60
GetAlarm			94	94	94	94	94	94
SetRelAlarm			100	100	100	100	100	100
SetAbsAlarm			112	112	112	112	112	112
CancelAlarm			82	82	82	82	82	82
InitCounter			70	70	70	70	70	70
GetCounterValue			82	82	82	82	82	82
osek_tick_alarm	<default>		80	80	80	80	80	80
	KL	2	46	46	46	46	46	46
osek_incr_counter			40	40	40	40	40	40
GetActiveApplicationMode		31	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			182	182	182	182	182	182

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
ShutdownOS	NoHook	12	50	50	50	50	50	50
	Hook	13	58	58	58	58	58	58
InitCOM			16	16	16	16	16	16
CloseCOM			16	16	16	16	16	16
StartCOM			42	42	42	42	42	42
StopCOM			32	32	32	32	32	32
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	68	68	68	160	160	160
	CCCB	15	160	160	160	160	160	160
GetMessageResource			62	62	62	62	62	62
ReleaseMessageResource			58	58	58	58	58	58
GetMessageStatus			50	50	50	50	50	50
SendMessage	SW CCCA	1, 14	88	88	88	190	190	190
	SW CCCB	1, 15	178	178	178	190	190	190
	NS CCCA	14	88	88	88	190	190	190
	NS CCCB	15	178	178	178	190	190	190
	KL CCCA	2, 14	66	66	66	166	166	166
	KL CCCB	2, 15	154	154	154	166	166	166
main_dispatch	NoHook	12	178	178	212	178	178	212
	Hook	13	216	216	250	216	216	250
sub_dispatch	B1LI	19	32	32	32	32	32	32
	B1LF	20	40	40	40	40	40	40
	B1HI	21	102	102	102	102	102	102
	B1HF	22	110	110	110	110	110	110
	B2LI	23	n/a	66	90	n/a	66	90
	B2LF	24	n/a	74	98	n/a	74	98
	B2HI	25	n/a	128	176	n/a	128	176
	B2HF	26	n/a	136	184	n/a	136	184
	E1HI	27	n/a	n/a	n/a	414	414	458
	E1HF	28	n/a	n/a	n/a	422	422	466
	E2HI	29	n/a	n/a	n/a	n/a	n/a	458
	E2HF	30	n/a	n/a	n/a	n/a	n/a	466
CAT2_wrapper			180	180	180	180	180	180
hook_support	No ServiceID	16	56	56	56	56	56	56
	No Parameters	17	64	64	64	64	64	64
		18	76	76	76	76	76	76
fp_support			80	80	80	80	80	80
utility_functions	common		96	96	96	96	96	96
	optional	32	n/a	n/a	n/a	n/a	n/a	n/a
	optional	32	74	74	74	74	74	74

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	88	88	88	88	88	88
Timing_termination		4	82	82	82	82	82	82
ActivateTaskset	SW	1	84	128	164	94	148	188
	NS		64	108	142	74	128	166
	KL	2	30	88	122	40	108	146
ChainTaskset	SWL	1, 8	66	110	144	66	128	158
	SWH	1, 9	100	146	182	100	156	196
	NSL	8	66	110	144	66	128	158
	NSH	9	88	134	170	88	144	184
GetTasksetRef			20	20	20	20	20	20
MergeTaskset			60	60	60	60	60	60
AssignTaskset			20	20	20	20	20	20
RemoveTaskset			60	60	60	60	60	60
TestSubTaskset			70	70	70	70	70	70
TestEquivalentTaskset			68	68	68	68	68	68
TickSchedule	SW	1	132	138	138	138	138	138
	NS		110	110	110	110	110	110
	KL	2	90	92	92	92	92	92
AdvanceSchedule	SW	1	126	126	126	126	126	126
	NS		104	98	98	98	98	98
	KL	2	82	80	80	80	80	80
StartSchedule			84	84	84	84	84	84
StopSchedule			68	68	68	68	68	68
GetScheduleStatus			96	96	96	96	96	96
GetScheduleValue			74	74	74	74	74	74
GetScheduleNext			22	22	22	22	22	22
SetScheduleNext			20	20	20	20	20	20
GetArrivalpointDelay			20	20	20	20	20	20
SetArrivalpointDelay			18	18	18	18	18	18
GetArrivalpointTasksetRef			18	18	18	18	18	18
GetArrivalpointNext			20	20	20	20	20	20
SetArrivalpointNext			18	18	18	18	18	18
TestArrivalpointWritable			42	42	42	42	42	42
GetExecutionTime			106	106	106	106	106	106
GetLargestExecutionTime			24	24	24	24	24	24
ResetLargestExecutionTime			24	24	24	24	24	24
GetStackOffset			40	40	40	40	40	40

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	176	216	248	188	226	278
	NS		208	248	280	220	258	308
	KL	2	148	188	220	160	198	248
TerminateTask	LExt	3	122	122	122	122	122	122
	H	5	144	144	144	144	144	144
ChainTask	SWL	1, 8	216	262	290	228	272	320
	SWH	1, 9	244	286	314	256	296	346
	NSL	8	256	302	330	268	312	360
	NSH	9	276	318	346	288	328	380
Schedule			182	182	202	182	182	202
GetTaskID			54	54	54	54	54	54
GetTaskState			182	182	182	188	188	188
EnableAllInterrupts			54	54	54	54	54	54
DisableAllInterrupts			62	62	62	62	62	62
ResumeAllInterrupts			92	92	92	92	92	92
SuspendAllInterrupts			80	80	80	80	80	80
ResumeOSInterrupts			92	92	92	92	92	92
SuspendOSInterrupts			80	80	80	80	80	80
GetResource	Task	7	254	254	230	254	254	230
	Combined	6	224	224	224	224	224	224
	CLEx	3	218	218	218	218	218	218
ReleaseResource	Task	7	236	236	236	236	236	236
	Combined	6	236	236	236	236	236	236
	CLEx	3	190	190	190	190	190	190
SetEvent	SW	1	n/a	n/a	n/a	226	226	298
	NS		n/a	n/a	n/a	252	252	324
	NS1i	10	n/a	n/a	n/a	156	n/a	n/a
	KL	2	n/a	n/a	n/a	198	198	270
	KL1i	2, 10	n/a	n/a	n/a	148	n/a	n/a
ClearEvent			n/a	n/a	n/a	122	122	122
GetEvent			n/a	n/a	n/a	174	174	174
WaitEvent	<default>		n/a	n/a	n/a	310	310	464
	fp	11	n/a	n/a	n/a	332	332	520
	1i	10	n/a	n/a	n/a	138	n/a	n/a
GetAlarmBase			118	118	118	118	118	118
GetAlarm			128	128	128	128	128	128
SetRelAlarm			160	160	160	160	160	160
SetAbsAlarm			170	170	170	170	170	170

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
CancelAlarm			116	116	116	116	116	116
InitCounter			130	130	130	130	130	130
GetCounterValue			140	140	140	140	140	140
osek_tick_alarm	<default>		80	80	80	80	80	80
	KL	2	46	46	46	46	46	46
osek_incr_counter			40	40	40	40	40	40
GetActiveApplicationMode		31	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			196	196	196	196	196	196
ShutdownOS	NoHook	12	60	60	60	60	60	60
	Hook	13	68	68	68	68	68	68
InitCOM			16	16	16	16	16	16
CloseCOM			16	16	16	16	16	16
StartCOM			56	56	56	56	56	56
StopCOM			54	54	54	54	54	54
ReadFlag			34	34	34	34	34	34
ResetFlag			36	36	36	36	36	36
ReceiveMessage	CCCA	14	120	120	120	210	210	210
	CCCB	15	210	210	210	210	210	210
GetMessageResource			96	96	96	96	96	96
ReleaseMessageResource			96	96	96	96	96	96
GetMessageStatus			98	98	98	98	98	98
SendMessage	SW CCCA	1, 14	146	146	146	246	246	246
	SW CCCB	1, 15	234	234	234	246	246	246
	NS CCCA	14	146	146	146	246	246	246
	NS CCCB	15	234	234	234	246	246	246
	KL CCCA	2, 14	142	142	142	242	242	242
	KL CCCB	2, 15	230	230	230	242	242	242
main_dispatch	NoHook	12	178	178	212	178	178	212
	Hook	13	216	216	250	216	216	250
sub_dispatch	B1LI	19	32	32	32	32	32	32
	B1LF	20	40	40	40	40	40	40
	B1HI	21	102	102	102	102	102	102
	B1HF	22	110	110	110	110	110	110
	B2LI	23	n/a	66	90	n/a	66	90
	B2LF	24	n/a	74	98	n/a	74	98
	B2HI	25	n/a	128	176	n/a	128	176
	B2HF	26	n/a	136	184	n/a	136	184
	E1HI	27	n/a	n/a	n/a	414	414	458
	E1HF	28	n/a	n/a	n/a	422	422	466
	E2HI	29	n/a	n/a	n/a	n/a	n/a	458
	E2HF	30	n/a	n/a	n/a	n/a	n/a	466

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
CAT2_wrapper			180	180	180	180	180	180
hook_support	No ServiceID	16	110	110	110	110	110	110
	No Parameters	17	122	122	122	122	122	122
		18	144	144	144	144	144	144
fp_support			80	80	80	80	80	80
utility_functions	common		96	96	96	96	96	96
	optional	32	n/a	n/a	n/a	n/a	n/a	n/a
	optional	32	74	74	74	74	74	74
validity_checks		3	38	38	38	38	38	38
Timing_dispatch		4	88	88	88	88	88	88
Timing_termination		4	82	82	82	82	82	82
ActivateTaskset	SW	1	238	290	320	260	316	372
	NS		262	314	344	284	340	396
	KL	2	204	256	288	226	282	338
ChainTaskset	SWL	1, 8	282	348	378	300	382	424
	SWH	1, 9	318	390	422	336	422	470
	NSL	8	326	394	428	346	434	480
	NSH	9	356	426	462	374	462	506
GetTasksetRef			118	118	118	118	118	118
MergeTaskset			188	188	188	188	188	188
AssignTaskset			160	160	160	160	160	160
RemoveTaskset			188	188	188	188	188	188
TestSubTaskset			186	186	186	186	186	186
TestEquivalentTaskset			184	184	184	184	184	184
TickSchedule	SW	1	260	206	206	206	206	206
	NS		286	252	252	252	252	252
	KL	2	236	180	180	180	180	180
AdvanceSchedule	SW	1	262	208	208	208	208	208
	NS		288	250	250	250	250	250
	KL	2	240	182	182	182	182	182
StartSchedule			168	168	168	168	168	168
StopSchedule			122	122	122	122	122	122
GetScheduleStatus			158	158	158	158	158	158
GetScheduleValue			134	134	134	134	134	134
GetScheduleNext			90	90	90	90	90	90
SetScheduleNext			154	154	154	154	154	154
GetArrivalpointDelay			116	116	116	116	116	116
SetArrivalpointDelay			138	138	138	138	138	138
GetArrivalpointTasksetRef			114	114	114	114	114	114
GetArrivalpointNext			116	116	116	116	116	116
SetArrivalpointNext			160	160	160	160	160	160

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
TestArrivalpointWritable			126	126	126	126	126	126
GetExecutionTime			120	120	120	120	120	120
GetLargestExecutionTime			94	94	94	94	94	94
ResetLargestExecutionTime			90	90	90	90	90	90
GetStackOffset			40	40	40	40	40	40

Notes

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

Number	Note
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

The collection of the performance data for the TMS470/TI 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 the OSEK COM class was set to CCCA for systems without events, CCCB for systems with events. `ShutdownOS()` enters a tight loop, and its execution time up to `ShutdownHook()`, when called, is measured.)

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	51	64	84	55	61	86
	NS	45	59	78	49	55	81
	KL	21	35	55	25	32	57
TerminateTask	LExt	0	0	0	0	0	0
	H	71	70	73	70	70	74
ChainTask	SWL	127	143	176	182	192	232
	SWH	159	174	207	215	221	262
	NSL	127	144	176	182	192	232
	NSH	156	171	204	211	217	259

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
Schedule	SW	49	49	55	48	48	55
GetTaskID		14	14	14	14	14	14
GetTaskState		49	50	49	55	55	55
EnableAllInterrupts		15	15	15	15	15	15
DisableAllInterrupts		24	24	24	24	24	24
ResumeAllInterrupts		20	20	20	20	20	20
SuspendAllInterrupts		31	31	30	30	31	31
ResumeOSInterrupts		20	20	20	20	20	20
SuspendOSInterrupts		30	30	31	31	30	30
GetResource	Task	18	18	19	18	18	19
	Combined	33	33	32	32	33	33
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	42	42	42	42	42	42
	Combined	56	56	56	56	56	56
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	59	59	59
	NS	n/a	n/a	n/a	56	56	56
	KL	n/a	n/a	n/a	33	32	33
ClearEvent		n/a	n/a	n/a	32	31	32
GetEvent		n/a	n/a	n/a	37	37	37
WaitEvent	<default>	n/a	n/a	n/a	230	230	270
	fp	n/a	n/a	n/a	234	234	273
GetAlarmBase		41	41	40	40	40	40
GetAlarm		50	50	50	50	50	50
SetRelAlarm		55	55	54	54	54	54
SetAbsAlarm		58	58	58	58	58	58
CancelAlarm		41	41	40	40	40	40
InitCounter		42	41	42	42	41	41
GetCounterValue		47	47	47	47	47	47
osek_tick_alarm	<default>	47	46	47	47	46	46
	KL	18	18	18	18	18	18
osek_incr_counter		6	5	6	6	5	5
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		509	509	509	509	509	509
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	29	29	28	29	29	29
InitCOM		3	4	3	4	3	3
CloseCOM		4	4	3	4	4	4
StartCOM		15	15	15	48	49	49
StopCOM		9	9	9	9	9	9
ReadFlag		n/a	n/a	n/a	7	7	7

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
ResetFlag		n/a	n/a	n/a	4	4	4
ReceiveMessage		38	38	38	132	132	133
GetMessageResource		n/a	n/a	n/a	44	43	44
ReleaseMessageResource		n/a	n/a	n/a	65	65	65
GetMessageStatus		n/a	n/a	n/a	18	18	18
SendMessage	SW	94	107	126	193	199	224
	NS	87	101	121	187	193	219
	KL	41	55	74	140	147	173
ActivateTaskset	SW	43	214	239	46	217	244
	NS	37	209	231	40	210	237
	KL	11	185	207	14	188	213
	SW2	43	215	239	46	216	244
	NS2	37	208	232	40	211	237
	KL2	11	186	207	14	187	213
ChainTaskset	SWL	123	294	332	175	347	385
	SWH	155	327	365	207	377	418
	NSL	123	294	331	175	347	385
	NSH	152	324	361	204	374	414
GetTasksetRef		11	11	10	11	11	10
MergeTaskset		38	38	39	38	38	39
AssignTaskset		9	9	9	9	9	9
RemoveTaskset		39	39	38	39	39	38
TestSubTaskset		43	43	42	43	43	42
TestEquivalentTaskset		42	42	42	42	42	42
TickSchedule	SW	70	255	276	84	262	286
	NS	62	246	267	75	253	276
	KL	39	224	245	53	231	254
	SW2	69	255	277	84	257	282
	NS2	62	246	267	75	248	273
	KL2	39	224	245	53	225	251
AdvanceSchedule	SW	63	245	266	74	252	275
	NS	55	236	257	65	243	267
	KL	31	214	235	42	221	244
	SW2	63	245	267	74	247	272
	NS2	54	236	257	65	238	263
	KL2	31	214	235	42	215	240
StartSchedule		53	53	53	53	53	53
StopSchedule		46	46	46	46	45	46
GetScheduleStatus		50	50	50	50	51	50
GetScheduleValue		48	48	48	48	47	48
GetScheduleNext		10	10	10	10	11	10
SetScheduleNext		10	10	10	10	11	10

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetArrivalpointDelay		9	9	9	9	9	9
SetArrivalpointDelay		9	9	9	9	9	9
GetArrivalpointTasksetRef		8	8	8	8	7	8
GetArrivalpointNext		9	9	9	9	9	9
SetArrivalpointNext		9	9	9	9	9	9
TestArrivalpointWritable		14	14	14	14	14	14
GetExecutionTime		5	5	5	5	5	5
GetLargestExecutionTime		8	8	8	8	8	8
ResetLargestExecutionTime		5	5	5	5	5	5
GetStackOffset		30	29	30	29	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	51	65	84	54	61	86
	NS	45	58	78	49	55	80
	KL	21	36	55	24	32	57
TerminateTask	LExt	0	0	0	0	0	0
	H	163	163	165	163	163	166
ChainTask	SWL	242	258	292	299	308	350
	SWH	262	277	311	319	325	368
	NSL	242	258	292	299	308	350
	NSH	258	273	307	316	321	364
Schedule	SW	49	48	55	49	49	55
GetTaskID		14	14	14	14	14	14
GetTaskState		50	49	50	55	55	55
EnableAllInterrupts		15	15	15	15	15	15
DisableAllInterrupts		24	24	24	24	24	24
ResumeAllInterrupts		20	20	20	20	20	20
SuspendAllInterrupts		31	31	30	30	31	31
ResumeOSInterrupts		20	20	20	20	20	20
SuspendOSInterrupts		30	30	31	31	30	30
GetResource	Task	18	18	19	18	18	19
	Combined	33	33	32	33	33	33
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	42	42	42	42	42	42
	Combined	59	59	59	59	59	59
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
SetEvent	SW	n/a	n/a	n/a	59	59	59
	NS	n/a	n/a	n/a	56	56	56
	KL	n/a	n/a	n/a	33	33	32
ClearEvent		n/a	n/a	n/a	32	32	31
GetEvent		n/a	n/a	n/a	37	37	37
WaitEvent	<default>	n/a	n/a	n/a	315	316	354
	fp	n/a	n/a	n/a	319	319	358
GetAlarmBase		41	40	41	40	40	40
GetAlarm		50	50	50	50	50	50
SetRelAlarm		55	54	55	54	54	54
SetAbsAlarm		58	58	58	58	58	58
CancelAlarm		41	40	41	40	40	40
InitCounter		41	41	41	41	42	41
GetCounterValue		47	47	47	47	47	47
osek_tick_alarm	<default>	46	46	46	46	47	46
	KL	18	18	18	18	18	18
osek_incr_counter		5	5	5	5	6	5
GetActiveApplicationMode		6	6	6	6	6	6
StartOS		1227	1227	1227	1227	1227	1227
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	29	29	28	29	29	29
InitCOM		4	3	3	4	4	3
CloseCOM		4	4	3	3	4	4
StartCOM		15	15	15	48	48	49
StopCOM		9	9	9	9	9	9
ReadFlag		n/a	n/a	n/a	7	7	7
ResetFlag		n/a	n/a	n/a	5	4	4
ReceiveMessage		37	37	38	132	133	133
GetMessageResource		n/a	n/a	n/a	43	43	43
ReleaseMessageResource		n/a	n/a	n/a	71	70	71
GetMessageStatus		n/a	n/a	n/a	17	18	17
SendMessage	SW	94	107	126	193	199	224
	NS	87	101	121	186	193	218
	KL	41	55	74	140	148	173
ActivateTaskset	SW	43	215	239	47	216	244
	NS	37	208	232	40	210	237
	KL	11	186	207	15	187	213
	SW2	43	214	239	46	216	245
	NS2	37	209	231	41	210	237
	KL2	11	185	207	14	187	212
ChainTaskset	SWL	238	409	448	292	464	503
	SWH	257	430	468	311	481	523
	NSL	238	409	448	292	464	503
	NSH	254	426	465	307	477	520

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetTasksetRef		10	11	11	11	11	10
MergeTaskset		39	38	38	38	38	39
AssignTaskset		9	9	9	9	9	9
RemoveTaskset		38	39	39	39	39	38
TestSubTaskset		42	43	43	43	43	42
TestEquivalentTaskset		42	42	42	42	42	42
TickSchedule	SW	69	255	276	84	262	286
	NS	62	246	268	75	253	277
	KL	39	224	245	53	230	254
	SW2	69	255	276	84	257	282
	NS2	62	246	268	75	248	273
	KL2	39	223	245	52	225	250
AdvanceSchedule	SW	63	245	266	74	252	275
	NS	54	236	258	65	243	266
	KL	31	214	235	43	220	244
	SW2	63	245	266	74	247	272
	NS2	55	236	258	65	238	263
	KL2	31	213	235	42	215	241
StartSchedule		53	53	53	53	53	53
StopSchedule		46	46	45	45	46	46
GetScheduleStatus		50	50	51	51	50	50
GetScheduleValue		48	48	47	47	48	48
GetScheduleNext		10	10	11	11	10	10
SetScheduleNext		10	10	11	11	10	10
GetArrivalpointDelay		9	9	9	9	9	9
SetArrivalpointDelay		9	9	9	9	9	9
GetArrivalpointTasksetRef		8	8	7	7	8	8
GetArrivalpointNext		9	9	9	9	9	9
SetArrivalpointNext		9	9	9	9	9	9
TestArrivalpointWritable		14	14	14	14	14	14
GetExecutionTime		59	59	59	59	59	59
GetLargestExecutionTime		12	13	13	12	12	13
ResetLargestExecutionTime		10	9	9	10	10	9
GetStackOffset		29	30	30	29	29	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	124	140	159	128	136	163
	NS	137	152	171	140	149	176
	KL	103	119	138	107	116	141
TerminateTask	LExt	194	194	197	195	195	198
	H	201	201	204	201	201	205
ChainTask	SWL	345	362	395	402	410	454
	SWH	362	380	413	420	428	471
	NSL	360	378	411	418	426	470
	NSH	374	392	426	432	441	483
Schedule	SW	65	66	72	66	65	72
GetTaskID		18	18	18	19	18	19
GetTaskState		134	134	134	137	137	137
EnableAllInterrupts		20	21	21	21	20	21
DisableAllInterrupts		29	30	30	30	29	30
ResumeAllInterrupts		30	30	30	30	30	30
SuspendAllInterrupts		36	36	36	36	36	36
ResumeOSInterrupts		30	30	30	30	30	30
SuspendOSInterrupts		36	36	36	36	36	36
GetResource	Task	193	193	121	211	211	139
	Combined	101	102	101	120	120	119
	CLEx	123	123	123	141	141	141
ReleaseResource	Task	111	110	111	128	128	129
	Combined	119	118	119	136	136	137
	CLEx	105	106	106	124	123	124
SetEvent	SW	n/a	n/a	n/a	142	142	142
	NS	n/a	n/a	n/a	149	149	149
	KL	n/a	n/a	n/a	124	123	123
ClearEvent		n/a	n/a	n/a	53	54	54
GetEvent		n/a	n/a	n/a	129	130	130
WaitEvent	<default>	n/a	n/a	n/a	363	363	396
	fp	n/a	n/a	n/a	367	367	399
GetAlarmBase		96	96	96	96	96	96
GetAlarm		103	103	103	103	103	103
SetRelAlarm		112	112	112	112	112	112
SetAbsAlarm		112	113	113	112	112	113
CancelAlarm		92	92	92	92	92	92
InitCounter		93	93	93	93	93	93
GetCounterValue		97	96	96	97	96	97
osek_tick_alarm	<default>	47	46	46	47	47	46
	KL	18	18	18	18	18	18

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
osek_incr_counter		6	5	5	6	6	5
GetActiveApplicationMode		7	7	6	7	6	6
StartOS		1277	1277	1277	1277	1277	1277
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	32	32	32	32	31	31
InitCOM		4	3	4	3	3	3
CloseCOM		4	3	4	4	3	3
StartCOM		24	23	24	55	55	55
StopCOM		17	16	17	17	16	16
ReadFlag		n/a	n/a	n/a	13	13	13
ResetFlag		n/a	n/a	n/a	11	11	11
ReceiveMessage		77	77	76	170	170	170
GetMessageResource		n/a	n/a	n/a	192	192	193
ReleaseMessageResource		n/a	n/a	n/a	188	187	188
GetMessageStatus		n/a	n/a	n/a	53	52	53
SendMessage	SW	205	220	239	302	310	336
	NS	217	232	252	314	323	350
	KL	169	185	204	268	277	302
ActivateTaskset	SW	243	422	441	251	422	454
	NS	251	431	449	259	431	463
	KL	219	401	419	228	402	433
	SW2	243	423	441	251	422	454
	NS2	251	430	449	259	431	463
	KL2	218	401	420	228	402	433
ChainTaskset	SWL	520	653	689	580	708	753
	SWH	534	667	704	593	720	769
	NSL	532	715	705	595	727	773
	NSH	545	680	767	604	736	782
GetTasksetRef		95	95	96	95	95	95
MergeTaskset		71	71	70	71	71	71
AssignTaskset		48	48	47	48	48	48
RemoveTaskset		71	71	70	71	71	71
TestSubTaskset		75	75	74	75	75	75
TestEquivalentTaskset		74	74	74	74	74	74
TickSchedule	SW	101	483	502	310	493	524
	NS	109	493	512	320	503	533
	KL	74	459	478	286	469	500
	SW2	101	483	501	310	484	516
	NS2	109	493	512	320	494	526
	KL2	74	459	477	286	460	492

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
AdvanceSchedule	SW	93	475	494	302	486	516
	NS	103	488	507	315	499	529
	KL	70	454	472	281	465	494
	SW2	93	475	494	302	477	508
	NS2	103	489	507	316	490	521
	KL2	70	454	472	281	455	487
StartSchedule		74	74	74	74	74	74
StopSchedule		59	59	59	59	60	59
GetScheduleStatus		65	65	65	65	64	65
GetScheduleValue		61	61	61	61	61	61
GetScheduleNext		28	28	28	28	27	28
SetScheduleNext		40	40	40	40	39	40
GetArrivalpointDelay		32	32	32	32	33	32
SetArrivalpointDelay		40	40	40	40	39	40
GetArrivalpointTasksetRef		31	31	31	31	31	31
GetArrivalpointNext		32	32	32	32	33	32
SetArrivalpointNext		45	45	45	45	45	45
TestArrivalpointWritable		36	36	36	36	35	36
GetExecutionTime		65	65	65	65	65	65
GetLargestExecutionTime		90	90	90	90	90	90
ResetLargestExecutionTime		84	84	84	84	84	84
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 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	28	28	28	28	28	28
	Cat 2	45	45	45	45	45	45

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	28	28	28	28	28	28
	Cat 2	94	94	94	95	95	95

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	28	28	28	28	28	28
	Cat 2	95	95	95	95	95	95

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.

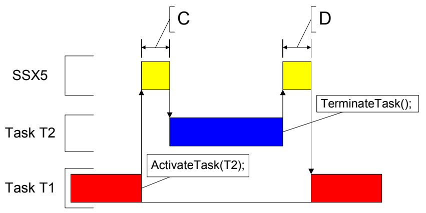

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

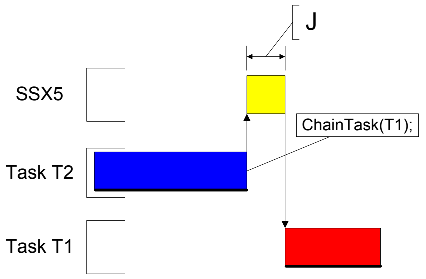

Figure 2: Task Chaining

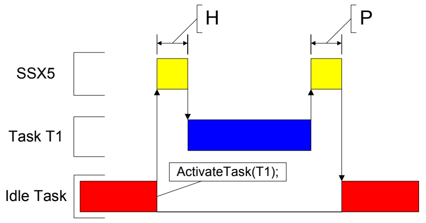

Figure 3: Task Activation from Idle Task

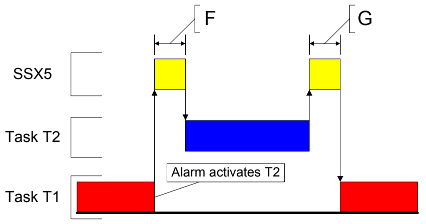

Figure 4: Task Activation from an Alarm

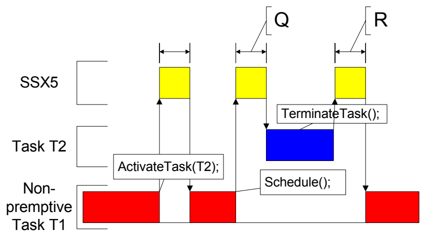

Figure 5: Non-Preemptive Task Calls Schedule()

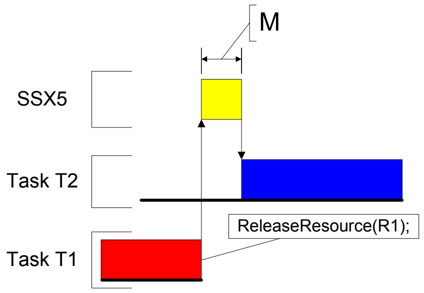

Figure 6: Blocked Task Activated by ReleaseResource()

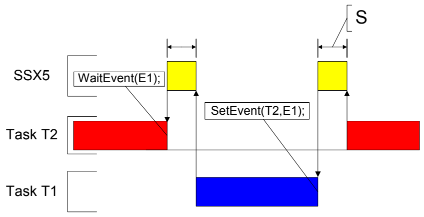

Figure 7: Waiting Task Activated by SetEvent()

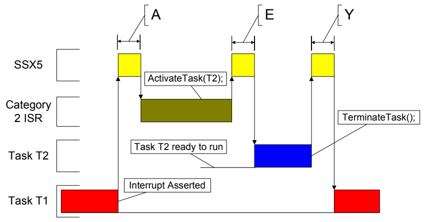

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	43	62	74	43	62	75
Figure 1: D	Heavy, Basic/Extended	72	89	101	96	96	108
ChainTask	Light, Basic	98	125	158	101	126	165
Figure 2: J	Heavy, Basic/Extended	219	262	306	246	269	320
Pre-emption	Light, Basic	94	119	158	98	119	163
Figure 1: C	Heavy, Basic/Extended	126	140	179	182	188	232
From idle task	Light, Basic	96	121	160	100	121	165
Figure 3: H	Heavy, Basic/Extended	128	142	181	184	190	234
Triggered by alarm	Light, Basic	158	183	221	161	183	227
Figure 4: F	Heavy, Basic/Extended	189	203	241	245	252	296
Schedule	Light, Basic	89	101	126	89	100	126
Figure 5: Q	Heavy, Basic/Extended	121	122	147	173	173	198
Release resource	Light, Basic	92	104	123	92	104	123
Figure 6: M	Heavy, Basic/Extended	124	125	144	176	176	196
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	286	286	356
From category 2 ISR	Light, Basic	66	78	97	66	78	97
Figure 8: E	Heavy, Basic/Extended	98	99	118	150	150	170

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	148	157	171	148	157	171
Figure 1: D	Heavy, Basic/Extended	165	173	185	180	181	193
ChainTask	Light, Basic	222	238	273	226	238	281
Figure 2: J	Heavy, Basic/Extended	423	447	493	443	454	507
Pre-emption	Light, Basic	156	169	212	159	169	217
Figure 1: C	Heavy, Basic/Extended	179	193	235	236	242	289
From idle task	Light, Basic	158	171	214	161	171	219
Figure 3: H	Heavy, Basic/Extended	181	195	236	238	244	291
Triggered by alarm	Light, Basic	219	233	275	223	233	281
Figure 4: F	Heavy, Basic/Extended	242	256	297	299	306	353
Schedule	Light, Basic	150	150	180	150	151	180
Figure 5: Q	Heavy, Basic/Extended	173	174	203	227	227	255

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	158	158	181	158	158	181
Figure 6: M	Heavy, Basic/Extended	181	182	204	235	234	257
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	322	322	396
From category 2 ISR	Light, Basic	232	231	254	231	231	255
Figure 8: E	Heavy, Basic/Extended	254	255	277	308	308	330

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	195	204	217	195	204	219
Figure 1: D	Heavy, Basic/Extended	203	211	223	218	218	231
ChainTask	Light, Basic	325	342	376	329	341	385
Figure 2: J	Heavy, Basic/Extended	561	587	634	582	595	649
Pre-emption	Light, Basic	225	241	283	229	241	288
Figure 1: C	Heavy, Basic/Extended	248	265	306	306	315	361
From idle task	Light, Basic	227	243	285	231	243	290
Figure 3: H	Heavy, Basic/Extended	250	266	307	308	316	363
Triggered by alarm	Light, Basic	288	304	346	292	304	351
Figure 4: F	Heavy, Basic/Extended	311	328	369	369	378	424
Schedule	Light, Basic	163	163	193	163	163	193
Figure 5: Q	Heavy, Basic/Extended	186	186	215	240	240	268
Release resource	Light, Basic	217	217	240	235	235	259
Figure 6: M	Heavy, Basic/Extended	240	241	263	312	312	334
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	421	421	484
From category 2 ISR	Light, Basic	237	237	260	237	237	260
Figure 8: E	Heavy, Basic/Extended	260	260	282	313	313	335

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	69	69	73	69	69	73
BCC1 lightweight, floating point	78	78	82	78	78	82
BCC1 heavyweight, integer	122	122	126	122	122	126
BCC1 heavyweight, floating point	122	122	126	122	122	126
BCC2 lightweight, integer	n/a	82	86	n/a	82	86
BCC2 lightweight, floating point	n/a	82	86	n/a	82	86
BCC2 heavyweight, integer	n/a	126	130	n/a	126	130
BCC2 heavyweight, floating point	n/a	126	130	n/a	126	130
ECC1 heavyweight, integer	n/a	n/a	n/a	186	186	190
ECC1 heavyweight, floating point	n/a	n/a	n/a	186	186	190
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	190
ECC2 heavyweight, floating point	n/a	n/a	n/a	n/a	n/a	190
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	77	77	77	77	77	77
BCC1 lightweight, floating point	86	86	86	86	86	86
BCC1 heavyweight, integer	130	130	130	130	130	130
BCC1 heavyweight, floating point	130	130	130	130	130	130
BCC2 lightweight, integer	n/a	90	90	n/a	90	90
BCC2 lightweight, floating point	n/a	90	90	n/a	90	90
BCC2 heavyweight, integer	n/a	134	134	n/a	134	134
BCC2 heavyweight, floating point	n/a	134	134	n/a	134	134
ECC1 heavyweight, integer	n/a	n/a	n/a	194	194	194
ECC1 heavyweight, floating point	n/a	n/a	n/a	194	194	194
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	194
ECC2 heavyweight, floating point	n/a	n/a	n/a	n/a	n/a	194

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		99	99	103	99	99	103
BCC1 lightweight, floating point		99	99	103	99	99	103
BCC1 heavyweight, integer		147	147	151	147	147	151
BCC1 heavyweight, floating point		147	147	151	147	147	151
BCC2 lightweight, integer		n/a	99	111	n/a	99	111
BCC2 lightweight, floating point		n/a	99	111	n/a	99	111
BCC2 heavyweight, integer		n/a	151	155	n/a	151	155
BCC2 heavyweight, floating point		n/a	151	155	n/a	151	155
ECC1 heavyweight, integer		n/a	n/a	n/a	211	211	215
ECC1 heavyweight, floating point		n/a	n/a	n/a	211	211	215
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	215
ECC2 heavyweight, floating point		n/a	n/a	n/a	n/a	n/a	215
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		107	107	107	107	107	107
BCC1 lightweight, floating point		107	107	107	107	107	107
BCC1 heavyweight, integer		155	155	155	155	155	155
BCC1 heavyweight, floating point		155	155	155	155	155	155
BCC2 lightweight, integer		n/a	107	115	n/a	107	115
BCC2 lightweight, floating point		n/a	107	115	n/a	107	115
BCC2 heavyweight, integer		n/a	159	159	n/a	159	159
BCC2 heavyweight, floating point		n/a	159	159	n/a	159	159
ECC1 heavyweight, integer		n/a	n/a	n/a	219	219	219
ECC1 heavyweight, floating point		n/a	n/a	n/a	219	219	219
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	219
ECC2 heavyweight, floating point		n/a	n/a	n/a	n/a	n/a	219

Extended

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
Pre- and Post-Task hooks not used							
Task type							
BCC1 lightweight, integer		99	99	103	99	99	103
BCC1 lightweight, floating point		99	99	103	99	99	103
BCC1 heavyweight, integer		147	147	151	147	147	151
BCC1 heavyweight, floating point		147	147	151	147	147	151
BCC2 lightweight, integer		n/a	99	111	n/a	99	111
BCC2 lightweight, floating point		n/a	99	111	n/a	99	111
BCC2 heavyweight, integer		n/a	151	155	n/a	151	155
BCC2 heavyweight, floating point		n/a	151	155	n/a	151	155
ECC1 heavyweight, integer		n/a	n/a	n/a	219	219	223
ECC1 heavyweight, floating point		n/a	n/a	n/a	219	219	223
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	223
ECC2 heavyweight, floating point		n/a	n/a	n/a	n/a	n/a	223
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		107	107	107	107	107	107
BCC1 lightweight, floating point		107	107	107	107	107	107
BCC1 heavyweight, integer		155	155	155	155	155	155
BCC1 heavyweight, floating point		155	155	155	155	155	155
BCC2 lightweight, integer		n/a	107	115	n/a	107	115
BCC2 lightweight, floating point		n/a	107	115	n/a	107	115
BCC2 heavyweight, integer		n/a	159	159	n/a	159	159
BCC2 heavyweight, floating point		n/a	159	159	n/a	159	159
ECC1 heavyweight, integer		n/a	n/a	n/a	227	227	227
ECC1 heavyweight, floating point		n/a	n/a	n/a	227	227	227
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	227
ECC2 heavyweight, floating point		n/a	n/a	n/a	n/a	n/a	227

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