LiveDevices

Binding Manual: Fujitsu 16LX

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

Version: RM00009-003   .01

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 16LX/FUJITSU 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.

The SSX5 libraries are built with the medium memory model. Application programs should also be built to use the medium memory model. If you wish to use any of the other three memory models you should contact LiveDevices.

2.1 Compiler

SSX5 was built using the following compiler:

Vendor	Fujitsu
Compiler	FFMC-16 Family Softune C Compiler
Version	V30L06 (Workbench version V30L26)

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

Option	Description
`-model MEDIUM`	Medium memory model

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

Option	Description
`-ilm[0-7]`	Set ILM level on entry to every function
`-regbank [0-31]`	Set the register bank to be used on entry to every function
`-except`	Automatic selection whether the system stack or the user stack is used
`-ramconst`	Specifies that the mirror function will not be used

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
`-model MEDIUM`	Medium memory model

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

Option	Description
`-ilm[0-7]`	Set ILM level on entry to every function
`-regbank [0-31]`	Set the register bank to be used on entry to every function
`-except`	Automatic selection whether the system stack or the user stack is used
`-ramconst`	Specifies that the mirror function will not be used

2.2 Assembler

SSX5 was built using the following assembler:

Vendor	Fujitsu
Assembler	FFMC-16 Family Softune Assembler
Version	V30L07 (Workbench version V30L26)

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

The compulsory linker/locator options for an SSX5 application are shown in the following table:

Option	Description
`-sc os_pird+os_pid/const/word=0xff4000`	Place `os_pird` and `os_pid` in “ROM mirror” section

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

The `os_pird` and `os_pid` sections must be placed in the “ROM mirror” section, so that they are visible to the 16-bit data pointers that access data in the medium memory model.

2.4 Debugger

Information about ORTI for RTA can be found in the *RTA ORTI Guide*.

At the time of writing, we were not aware of any debuggers for the 16LX with support for ORTI.

If you are using an ORTI version 2.0 aware debugger on this platform you can use the “unknown ORTI debugger” option in RTArchitect to generate an ORTI output file. The ORTI generated will not have been tested on the debugger and, therefore, is not guaranteed to work.

Please contact LiveDevices if ORTI fails to work correctly.

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

In SSX5 interrupts are allocated an Interrupt Priority Level (IPL). This is a processor independent abstraction of the interrupt priorities that are available on the target hardware. You can find out more about IPLs in the *RTA User Guide*. The hardware interrupt controller is explained in the Fujitsu hardware reference manuals.

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

Interrupt Priority Level (IPL)	**ILM Register**	**I Bit in PS Register**	**Description**
0	7	1	User level
1	6	1	Category 1 and 2 interrupts
2	5	1	Category 1 and 2 interrupts
3	4	1	Category 1 and 2 interrupts
4	3	1	Category 1 and 2 interrupts
5	2	1	Category 1 and 2 interrupts
6	1	1	Category 1 and 2 interrupts
7	0	1	Category 1 and 2 interrupts
8	any	0	Category 1 software interrupt only

3.1.2 Interrupt Vectors

On the Fujitsu 16LX, vectors are aligned on four byte boundaries between 0xFFFC00 and 0xFFFFFC. RTA allows ISRs to be bound to any vector, subject to the restrictions on ISRs described in Section 3.1.3.

Important: Extended intelligent I/O (see Fujitsu hardware documentation) should only be used with Category 1 ISRs.

3.1.3 Interrupt Priority Levels

The priority at which a hardware interrupt is taken is set in the ICR hardware registers. Each ICR register applies to two peripheral devices (for instance, on the MB90F548G chip, ICR03 sets the priority for the “16-bit reload timer 0” interrupt and for the “A/D converter” interrupt). This means that two devices, attached to a single ICR, share a hardware interrupt priority level.

RTArchitect generates a table, called `os_InitIrqLevels`, which must be used to initialize the ICR registers. This table contains the priority levels for interrupts defined in the application.

Important: The `os_InitIrqLevels` table must be copied to the ICR registers before the call to `StartOS()` otherwise interrupts will not work correctly.

The `init_target()` function in `target.c` in the example application (located in `<RTA3 install directory>\FUJI16LX\Example\`) gives an example of how to copy `os_InitIrqLevels` to the correct location.

ICR sharing by interrupt sources has ramifications with respect to interrupt sources that are not explicitly bound to an ISR.

When one of the interrupt sources on an ICR is bound to an ISR and the other is not, the priority of the unbound source is forced by the hardware to be the same as the bound one.

When neither interrupt source on an ICR is bound to an ISR, the value in the ICR is set to effectively disable the interrupts.

Important: If a default interrupt shares an ICR with another ISR, then only that default interrupt will trigger at the level of the other ISR value.

All software interrupts must be Category 1 and priority (IPL) 8. Vectors that can be used for peripheral interrupt sources can also be used for software interrupts. However, for a software interrupt, the priority in the ICR corresponding to that vector is meaningless. In the case where a peripheral interrupt source and a software interrupt have vectors that share the same ICR, it is, therefore, permitted to have their ISRs at different priorities.

If Category 1 interrupts are triggered from peripheral interrupt sources, they must have a priority (IPL) between 1 and 7 (where 1 is the lowest and 7 is the highest). All Category 1 ISRs must have a priority greater than or equal to that of the highest Category 2 interrupt.

Important: If you define a Category 1 interrupt at level 8, you must never trigger the interrupt using a hardware source.

Category 2 interrupts can have priorities (IPLs) between 1 and 7 (where 1 is the lowest and 7 is the highest).

3.1.4 Category 1 Handlers

Category 1 interrupt service routines (ISRs) must correctly handle the interrupt context themselves, without support from the operating system. The Fujitsu Softune 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.

3.1.5 Category 2 Handlers

Category 2 ISRs are provided with a C function context by SSX5, since SSX5 handles the interrupt context itself. The handlers are written using the OSEK OS standard `ISR()` macro, shown in Code Example 3:1.

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

Code Example 3:1 - Category 2 ISR Interrupt Handler

You must not insert a return from interrupt instruction in such a function. The return is handled automatically by SSX5.

3.1.6 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 includes 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 (`VVVVVV` represents the 6 hex digit, upper-case, zero-padded value of the vector location).

Vector Location	Label
`0xVVVVVV`	`_os_wrapper_VVVVVV`
eg :`0xFFFF6C`	`_os_wrapper_FFFF6C`

3.1.7 Automatic Vector Table Generation

The build process will generate a vector table covering all Category 1 and Category 2 ISRs defined in RTArchitect.

The reset vector (at address 0xFFFFDC) is set to the label `_start`.

If a default interrupt is specified, a vector table covering all vectors will be generated. If the default interrupt is not specified, a vector table will be generated that starts at the lowest used vector.

3.1.8 Manual Vector Table Generation

For each configured ISR, its associated vector must be programmed with the address of its handler. Other vectors may be programmed with the address of a default interrupt handler, if present.

The following example will place the address of `os_wrapper_FFFF6C` on interrupt vector number 36.

```
#pragma intvect os_wrapper_FFFF6C 36
```

3.2 Register Settings

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

Register	Required Value
SSP	Start of the stack

SSX5 uses the following hardware register. It should not be altered by user code.

Reserved Registers and Bits	Notes
PS	The Processor Status register (including the ILM, the RP and the CCR) should not be altered directly by user code.

Note: Instructions that indirectly change the condition codes in the CCR can, of course, be used freely.

SSX5 only uses register bank zero (i.e. RP=0 in PS) and this should not be altered by user code. Additional register bank memory can be used for other application purposes.

3.3 Stack Usage

3.3.1 Number of Stacks

RTA uses only the System stack and expects the user code to do the same.

The first argument to `StackFaultHook` is always 0.

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

Important: RTA requires a label "`_SStack_top`" marking the top of the stack. An example of how to place this label can be found in `start.asm` in the example application.

3.3.2 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): 36

Timing

API max usage (bytes): 36

Extended

API max usage (bytes): 44

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 Timing Issues

For timing to be correct and reproducible, all code alignment must be set to 2. However, the compiler outputs alignment of code set to 1 and there is no compiler option to make the compiler output with alignment of 2.

As a result, you must edit the intermediate assembly to change all `ALIGN=1` directives to `ALIGN=2`.

The ETCExample shows how this can be achieved automatically using the 'sed' stream editor (see `align_code.bat` in `<RTA3 install directory>\FUJI16LX\ETCExample`).

If you do not have 'sed' or any other stream editor you will need to pause the build process when an intermediate assembly file is produced and hand-edit the files to change `ALIGN=1` to `ALIGN=2`.

3.4.1 ETC and TCL Example

For the ETC and TCL Examples to measure correct timings, the code alignment must also be set to 2. In the distributed code for the ETC and TCL Examples, this is achieved using the stream editor 'sed', as described in Section 3.4.

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 six 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	**Fujitsu 16LX/MB90F548G**
Clock speed (MHz)	16
Code memory	On-chip FLASH
Read-only data memory	On-chip FLASH
Read-write data memory	On-chip RAM

4.1 Functionality

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

Configuration	**Application Uses**					
Events	**No**			**Yes**		
Shared Task Priorities	**No**		**Yes**	**No**		**Yes**
Multiple Task Activations	**No**	**Yes**		**No**	**Yes**	
Maximum number of tasks	16	16	16	16	16	16
Maximum number of not suspended tasks	16	16	16	16	16	16
Maximum number of priorities	16	16	16	16	16	16
Number of tasks per priority (for BCC2 and ECC2)	n/a	16	16	n/a	16	16
Upper limit for number of basic task activations per task priority	1	255	255	1	255	255
Maximum number of events per task	0	0	0	16	16	16
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		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
OS overhead	RAM	12	12	12	12	12	12
	ROM	112	112	112	112	112	112
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Timing

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
OS overhead	RAM	22	22	22	22	22	22
	ROM	148	148	148	148	148	148
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

Extended

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
OS overhead	RAM	29	29	29	29	29	29
	ROM	168	168	168	168	168	168
COM overhead	RAM	2	2	2	2	2	2
	ROM	5	5	5	5	5	5

4.2.2 ROM and RAM for OSEK OS Objects

In addition to the base OS overhead, detailed in Section 4.2.1, each OSEK OS object requires ROM and/or RAM. SSX5 provides additional sub-task types for each task type in OSEK (basic and extended), determined by the offline configuration tools. They are as follows:

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

The following tables give the ROM and/or RAM requirements (in bytes) for each OS object in SSX5. (Note that the OSEK COM class was set to CCCA for systems without events, CCCB for systems with events. A default message of size 10 bytes was used for both CCCA and CCCB. The CCCB message size includes queued messages.)

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	0	0	0	0	0	0
	ROM	24	24	24	24	24	24
BCC1 Heavyweight task	RAM	2	2	2	2	2	2
	ROM	26	26	26	26	26	26
BCC2 task	RAM	n/a	4	6	n/a	4	6
	ROM	n/a	28	32	n/a	28	32
ECC1, Integer task	RAM	n/a	n/a	n/a	26	26	26
	ROM	n/a	n/a	n/a	36	36	36
ECC1, floating-point task	RAM	n/a	n/a	n/a	27	27	27
	ROM	n/a	n/a	n/a	36	36	36
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	28
	ROM	n/a	n/a	n/a	n/a	n/a	40
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	29
	ROM	n/a	n/a	n/a	n/a	n/a	40
Category 2 ISR	RAM	0	0	0	0	0	0
	ROM	31	31	31	31	31	31

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Category 2 ISR, floating-point	RAM	1	1	1	1	1	1
	ROM	44	44	44	44	44	44
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Alarm	RAM	6	6	6	6	6	6
	ROM	30	30	30	30	30	30
Counter	RAM	2	2	2	2	2	2
	ROM	26	26	26	26	26	26
Message	RAM	11	11	11	31	31	31
	ROM	12	12	12	30	30	30
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	6	0	6	6
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	8	8	8	8	8	8
	ROM	26	26	26	26	26	26
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

Timing

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	6	6	6	6	6	6
	ROM	30	30	30	30	30	30
BCC1 Heavyweight task	RAM	8	8	8	8	8	8
	ROM	32	32	32	32	32	32
BCC2 task	RAM	n/a	10	12	n/a	10	12
	ROM	n/a	34	38	n/a	34	38
ECC1, Integer task	RAM	n/a	n/a	n/a	32	32	32
	ROM	n/a	n/a	n/a	42	42	42
ECC1, floating-point task	RAM	n/a	n/a	n/a	33	33	33
	ROM	n/a	n/a	n/a	42	42	42
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	34
	ROM	n/a	n/a	n/a	n/a	n/a	46
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	35
	ROM	n/a	n/a	n/a	n/a	n/a	46
Category 2 ISR	RAM	6	6	6	6	6	6
	ROM	50	50	50	50	50	50
Category 2 ISR, floating-point	RAM	7	7	7	7	7	7
	ROM	58	58	58	58	58	58
Resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0
Linked resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Alarm	RAM	6	6	6	6	6	6
	ROM	30	30	30	30	30	30
Counter	RAM	2	2	2	2	2	2
	ROM	26	26	26	26	26	26
Message	RAM	11	11	11	31	31	31
	ROM	12	12	12	30	30	30
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	6	0	6	6

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	6	6	6	6	6	6
Arrivalpoint (writable)	RAM	6	6	6	6	6	6
	ROM	6	6	6	6	6	6
Schedule	RAM	8	8	8	8	8	8
	ROM	26	26	26	26	26	26
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
BCC1 Lightweight task	RAM	7	7	7	7	7	7
	ROM	34	34	34	34	34	34
BCC1 Heavyweight task	RAM	10	10	10	10	10	10
	ROM	34	34	34	34	34	34
BCC2 task	RAM	n/a	12	14	n/a	12	14
	ROM	n/a	36	40	n/a	36	40
ECC1, Integer task	RAM	n/a	n/a	n/a	34	34	34
	ROM	n/a	n/a	n/a	44	44	44
ECC1, floating-point task	RAM	n/a	n/a	n/a	35	35	35
	ROM	n/a	n/a	n/a	44	44	44
ECC2, Integer task	RAM	n/a	n/a	n/a	n/a	n/a	36
	ROM	n/a	n/a	n/a	n/a	n/a	48
ECC2, floating-point task	RAM	n/a	n/a	n/a	n/a	n/a	37
	ROM	n/a	n/a	n/a	n/a	n/a	48
Category 2 ISR	RAM	7	7	7	7	7	7
	ROM	54	54	54	54	54	54
Category 2 ISR, floating-point	RAM	8	8	8	8	8	8
	ROM	62	62	62	62	62	62
Resource	RAM	4	4	4	4	4	4
	ROM	14	14	14	14	14	14
Internal resource	RAM	0	0	0	0	0	0
	ROM	0	0	0	0	0	0

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Linked resource	RAM	4	4	4	4	4	4
	ROM	14	14	14	14	14	14
Alarm	RAM	6	6	6	6	6	6
	ROM	32	32	32	32	32	32
Counter	RAM	2	2	2	2	2	2
	ROM	28	28	28	28	28	28
Message	RAM	11	11	11	31	31	31
	ROM	14	14	14	32	32	32
Flag	RAM	1	1	1	1	1	1
	ROM	1	1	1	1	1	1
Message resource	RAM	4	4	4	4	4	4
	ROM	14	14	14	14	14	14
Event	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Priority level	RAM	0	0	4	0	4	4
	ROM	0	0	6	0	6	6
Arrivalpoint (readonly)	RAM	0	0	0	0	0	0
	ROM	10	10	10	10	10	10
Arrivalpoint (writable)	RAM	10	10	10	10	10	10
	ROM	10	10	10	10	10	10
Schedule	RAM	10	10	10	10	10	10
	ROM	30	30	30	30	30	30
Taskset (readonly)	RAM	0	0	0	0	0	0
	ROM	2	2	2	2	2	2
Taskset (writable)	RAM	2	2	2	2	2	2
	ROM	2	2	2	2	2	2

4.2.3 Size of Linkable Modules

SSX5 is demand linked. This means that each API call is placed into a separately linkable module. The following sections list the module sizes (in bytes) for each API call in the 3 SSX5 OS status types (standard, timing, and extended).

In some cases there are multiple variants of particular API calls. This is because the offline configuration of SSX5 can determine when optimized versions of the API calls can be used. The smallest and fastest call will be selected. In these cases, modules sizes are given for each variant under the particular configuration of SSX5 for which the call is valid.

The call variants are as follows:

Variant	Description
1i	Idle task is only ECC task.
CCCA	OSEK COM class.
CCCB	OSEK COM class.
CLEx	Resource tests in Extended OS Status.
fp	ECC task uses floating-point.
H	Used for heavyweight termination only.
Hook	Pre- and Post- Task hooks are used.
KL	API is called from OS level.
KL1i	API is called from OS level, idle task is only ECC task.
KL2	Activated taskset has one BCC2 task.
LExt	Used for lightweight termination in Extended Status.
ServiceID	`ErrorHook` uses `GetServiceID`, but does not use `GetServiceParameters`.
Parameters	`ErrorHook` uses `GetServiceID` and `GetServiceParameters`.
NoHook	Pre- and/or Post- Task hooks are not used.
NS	No context switch is possible.
NS1i	No context switch is possible, idle task is only ECC task.
NS2	Activated taskset has one BCC2 task.
NSH	Chain from heavyweight task, not to higher priority.
NSL	Chain from lightweight task, not to higher priority.
Shared	Resource is used by tasks and ISRs.
SW	A context switch is made if required.
SW2	Activated taskset has one BCC2 task.
SWH	Chain from heavyweight task to possibly higher priority.
SWL	Chain from lightweight task to possibly higher priority.
Task	Resource is used only by tasks.

Standard

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
Service name	Variant	Notes						
ActivateTask	SW	1	92	145	195	96	149	214
	NS		73	126	176	77	130	195
	KL	2	53	107	157	57	111	176
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	22	22	22	22	22	22
ChainTask	SWL	1, 8	73	128	178	77	132	197
	SWH	1, 9	97	150	201	101	154	220
	NSL	8	73	128	178	77	132	197
	NSH	9	91	144	195	95	148	214

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
Schedule			55	55	79	55	55	79
GetTaskID			21	21	21	21	21	21
GetTaskState			57	57	57	72	72	72
EnableAllInterrupts			10	10	10	10	10	10
DisableAllInterrupts			10	10	10	10	10	10
ResumeAllInterrupts			26	26	26	26	26	26
SuspendAllInterrupts			21	21	21	21	21	21
ResumeOSInterrupts			28	28	28	28	28	28
SuspendOSInterrupts			36	36	36	36	36	36
GetResource	Task	7	26	26	30	26	26	30
	Combined	6	63	63	63	63	63	63
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	49	49	49	49	49	49
	Combined	6	101	101	101	101	101	101
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	84	84	173
	NS		n/a	n/a	n/a	65	65	151
	NS1i	10	n/a	n/a	n/a	33	n/a	n/a
	KL	2	n/a	n/a	n/a	52	52	139
	KL1i	2, 10	n/a	n/a	n/a	17	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	17	17	17
WaitEvent	<default>		n/a	n/a	n/a	191	191	365
	fp	11	n/a	n/a	n/a	223	223	432
	1i	10	n/a	n/a	n/a	18	n/a	n/a
GetAlarmBase			41	41	41	41	41	41
GetAlarm			79	79	79	79	79	79
SetRelAlarm			84	84	84	84	84	84
SetAbsAlarm			108	108	108	108	108	108
CancelAlarm			66	66	66	66	66	66
InitCounter			43	43	43	43	43	43
GetCounterValue			50	50	50	50	50	50
osek_tick_alarm	<default>		51	51	51	51	51	51
	KL	2	38	38	38	38	38	38
osek_incr_counter			39	39	39	39	39	39
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			88	88	88	88	88	88
ShutdownOS	NoHook	12	17	17	17	17	17	17
	Hook	13	29	29	29	29	29	29
InitCOM			2	2	2	2	2	2

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
CloseCOM			2	2	2	2	2	2
StartCOM			17	17	17	17	17	17
StopCOM			12	12	12	12	12	12
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	46	46	46	153	153	153
	CCCB	15	153	153	153	153	153	153
GetMessageResource			34	34	34	34	34	34
ReleaseMessageResource			34	34	34	34	34	34
GetMessageStatus			51	51	51	51	51	51
SendMessage	SW CCCA	1, 14	65	65	65	179	179	179
	SW CCCB	1, 15	163	163	163	179	179	179
	NS CCCA	14	65	65	65	179	179	179
	NS CCCB	15	163	163	163	179	179	179
	KL CCCA	2, 14	53	53	53	167	167	167
	KL CCCB	2, 15	151	151	151	167	167	167
main_dispatch	NoHook	12	75	75	107	75	75	107
	Hook	13	106	106	138	106	106	138
sub_dispatch	B1LF	19	24	24	24	24	24	24
	B1HI	20	66	66	66	66	66	66
	B1HF	21	74	74	74	74	74	74
	B2LI	22	n/a	51	85	n/a	51	85
	B2LF	23	n/a	59	93	n/a	59	93
	B2HI	24	n/a	102	198	n/a	102	198
	B2HF	25	n/a	110	206	n/a	110	206
	E1HI	26	n/a	n/a	n/a	280	280	378
	E1HF	27	n/a	n/a	n/a	288	288	386
	E2HI	28	n/a	n/a	n/a	n/a	n/a	378
	E2HF	29	n/a	n/a	n/a	n/a	n/a	386
ErrorHook support		16	26	26	26	26	26	26
	ServiceID	17	32	32	32	32	32	32
	Parameters	18	53	53	53	53	53	53
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	48	85	151	53	97	170
	NS		29	66	132	34	78	151
	KL	2	13	54	116	22	66	135
ChainTaskset	SWL	1, 8	33	73	135	33	79	148
	SWH	1, 9	67	108	174	67	114	187
	NSL	8	33	73	135	33	79	148
	NSH	9	61	102	168	61	108	181

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
GetTasksetRef			13	13	13	13	13	13
MergeTaskset			30	30	30	30	30	30
AssignTaskset			13	13	13	13	13	13
RemoveTaskset			31	31	31	31	31	31
TestSubTaskset			48	48	48	48	48	48
TestEquivalentTaskset			42	42	42	42	42	42
TickSchedule	SW	1	110	110	110	110	110	110
	NS		91	91	91	91	91	91
	KL	2	76	76	76	76	76	76
AdvanceSchedule	SW	1	104	104	104	104	104	104
	NS		85	85	85	85	85	85
	KL	2	70	70	70	70	70	70
StartSchedule			62	62	62	62	62	62
StopSchedule			42	42	42	42	42	42
GetScheduleStatus			72	72	72	72	72	72
GetScheduleValue			50	50	50	50	50	50
GetScheduleNext			15	15	15	15	15	15
SetScheduleNext			18	18	18	18	18	18
GetArrivalpointDelay			14	14	14	14	14	14
SetArrivalpointDelay			12	12	12	12	12	12
GetArrivalpointTasksetRef			11	11	11	11	11	11
GetArrivalpointNext			14	14	14	14	14	14
SetArrivalpointNext			12	12	12	12	12	12
TestArrivalpointWritable			26	26	26	26	26	26
GetExecutionTime			2	2	2	2	2	2
GetLargestExecutionTime			10	10	10	10	10	10
ResetLargestExecutionTime			2	2	2	2	2	2
GetStackOffset			19	19	19	19	19	19
Interrupt support			15	15	15	15	15	15
Utility functions			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	
Service name	Variant	Notes						
ActivateTask	SW	1	92	145	195	96	149	214
	NS		73	126	176	77	130	195
	KL	2	53	107	157	57	111	176
TerminateTask	LExt	3	n/a	n/a	n/a	n/a	n/a	n/a
	H	5	22	22	22	22	22	22
ChainTask	SWL	1, 8	73	128	178	77	132	197
	SWH	1, 9	97	150	201	101	154	220
	NSL	8	73	128	178	77	132	197
	NSH	9	91	144	195	95	148	214
Schedule			62	62	86	62	62	86
GetTaskID			21	21	21	21	21	21
GetTaskState			57	57	57	72	72	72
EnableAllInterrupts			10	10	10	10	10	10
DisableAllInterrupts			10	10	10	10	10	10
ResumeAllInterrupts			26	26	26	26	26	26
SuspendAllInterrupts			21	21	21	21	21	21
ResumeOSInterrupts			28	28	28	28	28	28
SuspendOSInterrupts			36	36	36	36	36	36
GetResource	Task	7	26	26	30	26	26	30
	Combined	6	63	63	63	63	63	63
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	7	56	56	56	56	56	56
	Combined	6	115	115	115	115	115	115
	CLEx	3	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	1	n/a	n/a	n/a	84	84	173
	NS		n/a	n/a	n/a	65	65	151
	NS1i	10	n/a	n/a	n/a	33	n/a	n/a
	KL	2	n/a	n/a	n/a	52	52	139
	KL1i	2, 10	n/a	n/a	n/a	17	n/a	n/a
ClearEvent			n/a	n/a	n/a	28	28	28
GetEvent			n/a	n/a	n/a	17	17	17
WaitEvent	<default>		n/a	n/a	n/a	229	229	403
	fp	11	n/a	n/a	n/a	261	261	470
	1i	10	n/a	n/a	n/a	60	n/a	n/a
GetAlarmBase			41	41	41	41	41	41
GetAlarm			79	79	79	79	79	79
SetRelAlarm			84	84	84	84	84	84

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
SetAbsAlarm			108	108	108	108	108	108
CancelAlarm			66	66	66	66	66	66
InitCounter			43	43	43	43	43	43
GetCounterValue			50	50	50	50	50	50
osek_tick_alarm	<default>		51	51	51	51	51	51
	KL	2	38	38	38	38	38	38
osek_incr_counter			39	39	39	39	39	39
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			120	120	120	120	120	120
ShutdownOS	NoHook	12	17	17	17	17	17	17
	Hook	13	29	29	29	29	29	29
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			17	17	17	17	17	17
StopCOM			12	12	12	12	12	12
ReadFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ResetFlag		30	n/a	n/a	n/a	n/a	n/a	n/a
ReceiveMessage	CCCA	14	46	46	46	153	153	153
	CCCB	15	153	153	153	153	153	153
GetMessageResource			34	34	34	34	34	34
ReleaseMessageResource			34	34	34	34	34	34
GetMessageStatus			51	51	51	51	51	51
SendMessage	SW CCCA	1, 14	65	65	65	179	179	179
	SW CCCB	1, 15	163	163	163	179	179	179
	NS CCCA	14	65	65	65	179	179	179
	NS CCCB	15	163	163	163	179	179	179
	KL CCCA	2, 14	53	53	53	167	167	167
	KL CCCB	2, 15	151	151	151	167	167	167
main_dispatch	NoHook	12	147	147	181	147	147	181
	Hook	13	171	171	213	171	171	213
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	77	77	77	77	77	77
	B1HF	21	85	85	85	85	85	85
	B2LI	22	n/a	40	76	n/a	40	76
	B2LF	23	n/a	48	84	n/a	48	84
	B2HI	24	n/a	108	201	n/a	108	201
	B2HF	25	n/a	116	209	n/a	116	209
	E1HI	26	n/a	n/a	n/a	304	304	402
	E1HF	27	n/a	n/a	n/a	312	312	410
	E2HI	28	n/a	n/a	n/a	n/a	n/a	402
	E2HF	29	n/a	n/a	n/a	n/a	n/a	410

Configuration			**Application Uses**					
Events			**No**			**Yes**		
Shared Task Priorities			**No**		**Yes**	**No**		**Yes**
Multiple Task Activations			**No**	**Yes**		**No**	**Yes**	
ErrorHook support		16	26	26	26	26	26	26
	ServiceID	17	32	32	32	32	32	32
	Parameters	18	53	53	53	53	53	53
validity_checks		3	n/a	n/a	n/a	n/a	n/a	n/a
Timing_dispatch		4	49	49	49	49	49	49
Timing_termination		4	92	92	92	92	92	92
ActivateTaskset	SW	1	48	85	151	53	97	170
	NS		29	66	132	34	78	151
	KL	2	13	54	116	22	66	135
ChainTaskset	SWL	1, 8	33	73	135	33	79	148
	SWH	1, 9	67	108	174	67	114	187
	NSL	8	33	73	135	33	79	148
	NSH	9	61	102	168	61	108	181
GetTasksetRef			13	13	13	13	13	13
MergeTaskset			30	30	30	30	30	30
AssignTaskset			13	13	13	13	13	13
RemoveTaskset			31	31	31	31	31	31
TestSubTaskset			48	48	48	48	48	48
TestEquivalentTaskset			42	42	42	42	42	42
TickSchedule	SW	1	110	110	110	110	110	110
	NS		91	91	91	91	91	91
	KL	2	76	76	76	76	76	76
AdvanceSchedule	SW	1	104	104	104	104	104	104
	NS		85	85	85	85	85	85
	KL	2	70	70	70	70	70	70
StartSchedule			62	62	62	62	62	62
StopSchedule			42	42	42	42	42	42
GetScheduleStatus			72	72	72	72	72	72
GetScheduleValue			50	50	50	50	50	50
GetScheduleNext			15	15	15	15	15	15
SetScheduleNext			18	18	18	18	18	18
GetArrivalpointDelay			14	14	14	14	14	14
SetArrivalpointDelay			12	12	12	12	12	12
GetArrivalpointTasksetRef			11	11	11	11	11	11
GetArrivalpointNext			14	14	14	14	14	14
SetArrivalpointNext			12	12	12	12	12	12
TestArrivalpointWritable			26	26	26	26	26	26
GetExecutionTime			56	56	56	56	56	56
GetLargestExecutionTime			22	22	22	22	22	22
ResetLargestExecutionTime			19	19	19	19	19	19
GetStackOffset			19	19	19	19	19	19
Interrupt support			15	15	15	15	15	15
Utility functions			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	
Service name	Variant	Notes						
ActivateTask	SW	1	177	228	280	183	232	299
	NS		227	284	335	233	288	354
	KL	2	124	179	232	130	183	251
TerminateTask	LExt	3	82	82	82	82	82	82
	H	5	107	107	107	107	107	107
ChainTask	SWL	1, 8	203	260	310	209	265	330
	SWH	1, 9	232	287	343	238	291	363
	NSL	8	271	330	380	278	334	399
	NSH	9	301	359	415	307	363	434
Schedule			153	153	179	153	153	179
GetTaskID			29	29	29	29	29	29
GetTaskState			175	175	175	181	181	181
EnableAllInterrupts			18	18	18	18	18	18
DisableAllInterrupts			20	20	20	20	20	20
ResumeAllInterrupts			63	63	63	63	63	63
SuspendAllInterrupts			31	31	31	31	31	31
ResumeOSInterrupts			65	65	65	65	65	65
SuspendOSInterrupts			46	46	46	46	46	46
GetResource	Task	7	310	310	270	310	310	270
	Combined	6	269	269	269	269	269	269
	CLEx	3	257	257	257	257	257	257
ReleaseResource	Task	7	270	270	270	270	270	270
	Combined	6	324	324	324	324	324	324
	CLEx	3	254	254	254	254	254	254
SetEvent	SW	1	n/a	n/a	n/a	214	214	309
	NS		n/a	n/a	n/a	270	270	364
	NS1i	10	n/a	n/a	n/a	162	n/a	n/a
	KL	2	n/a	n/a	n/a	165	165	255
	KL1i	2, 10	n/a	n/a	n/a	129	n/a	n/a
ClearEvent			n/a	n/a	n/a	93	93	93
GetEvent			n/a	n/a	n/a	124	124	124
WaitEvent	<default>		n/a	n/a	n/a	323	323	483
	fp	11	n/a	n/a	n/a	355	355	550
	1i	10	n/a	n/a	n/a	153	n/a	n/a
GetAlarmBase			130	130	130	130	130	130
GetAlarm			134	134	134	134	134	134
SetRelAlarm			176	176	176	176	176	176

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
SetAbsAlarm			199	199	199	199	199	199
CancelAlarm			119	119	119	119	119	119
InitCounter			134	134	134	134	134	134
GetCounterValue			144	144	144	144	144	144
osek_tick_alarm	<default>		77	77	77	77	77	77
	KL	2	38	38	38	38	38	38
osek_incr_counter			39	39	39	39	39	39
GetActiveApplicationMode		30	n/a	n/a	n/a	n/a	n/a	n/a
StartOS			130	130	130	130	130	130
ShutdownOS	NoHook	12	22	22	22	22	22	22
	Hook	13	34	34	34	34	34	34
InitCOM			2	2	2	2	2	2
CloseCOM			2	2	2	2	2	2
StartCOM			27	27	27	27	27	27
StopCOM			28	28	28	28	28	28
ReadFlag			22	22	22	22	22	22
ResetFlag			29	29	29	29	29	29
ReceiveMessage	CCCA	14	117	117	117	220	220	220
	CCCB	15	220	220	220	220	220	220
GetMessageResource			69	69	69	69	69	69
ReleaseMessageResource			69	69	69	69	69	69
GetMessageStatus			85	85	85	85	85	85
SendMessage	SW CCCA	1, 14	139	139	139	252	252	252
	SW CCCB	1, 15	236	236	236	252	252	252
	NS CCCA	14	139	139	139	252	252	252
	NS CCCB	15	236	236	236	252	252	252
	KL CCCA	2, 14	110	110	110	225	225	225
	KL CCCB	2, 15	209	209	209	225	225	225
main_dispatch	NoHook	12	147	147	181	147	147	181
	Hook	13	171	171	213	171	171	213
sub_dispatch	B1LF	19	13	13	13	13	13	13
	B1HI	20	78	78	78	78	78	78
	B1HF	21	86	86	86	86	86	86
	B2LI	22	n/a	40	76	n/a	40	76
	B2LF	23	n/a	48	84	n/a	48	84
	B2HI	24	n/a	109	202	n/a	109	202
	B2HF	25	n/a	117	210	n/a	117	210
	E1HI	26	n/a	n/a	n/a	305	305	403
	E1HF	27	n/a	n/a	n/a	313	313	411
	E2HI	28	n/a	n/a	n/a	n/a	n/a	403
	E2HF	29	n/a	n/a	n/a	n/a	n/a	411

Configuration			Application Uses					
Events			No			Yes		
Shared Task Priorities			No		Yes	No		Yes
Multiple Task Activations			No	Yes		No	Yes	
ErrorHook support		16	72	72	72	72	72	72
	ServiceID	17	78	78	78	78	78	78
	Parameters	18	99	99	99	99	99	99
validity_checks		3	30	30	30	30	30	30
Timing_dispatch		4	49	49	49	49	49	49
Timing_termination		4	92	92	92	92	92	92
ActivateTaskset	SW	1	244	286	360	254	302	389
	NS		297	338	411	307	354	440
	KL	2	195	237	307	205	254	336
ChainTaskset	SWL	1, 8	294	337	403	302	348	427
	SWH	1, 9	334	388	454	342	399	477
	NSL	8	364	407	473	372	418	497
	NSH	9	398	452	518	406	463	541
GetTasksetRef			104	104	104	104	104	104
MergeTaskset			205	205	205	205	205	205
AssignTaskset			159	159	159	159	159	159
RemoveTaskset			206	206	206	206	206	206
TestSubTaskset			220	220	220	220	220	220
TestEquivalentTaskset			218	218	218	218	218	218
TickSchedule	SW	1	268	206	206	206	206	206
	NS		333	299	299	299	299	299
	KL	2	223	164	164	164	164	164
AdvanceSchedule	SW	1	282	218	218	218	218	218
	NS		344	305	305	305	305	305
	KL	2	247	183	183	183	183	183
StartSchedule			186	186	186	186	186	186
StopSchedule			138	138	138	138	138	138
GetScheduleStatus			173	173	173	173	173	173
GetScheduleValue			141	141	141	141	141	141
GetScheduleNext			85	85	85	85	85	85
SetScheduleNext			150	150	150	150	150	150
GetArrivalpointDelay			112	112	112	112	112	112
SetArrivalpointDelay			126	126	126	126	126	126
GetArrivalpointTasksetRef			111	111	111	111	111	111
GetArrivalpointNext			112	112	112	112	112	112
SetArrivalpointNext			174	174	174	174	174	174
TestArrivalpointWritable			126	126	126	126	126	126
GetExecutionTime			92	92	92	92	92	92
GetLargestExecutionTime			89	89	89	89	89	89
ResetLargestExecutionTime			83	83	83	83	83	83
GetStackOffset			19	19	19	19	19	19
Interrupt support			15	15	15	15	15	15
Utility functions			16	16	16	16	16	16

Notes

Number	Note
1	Linked only if upward activations are allowed
2	Linked only if API is called within ISR
3	Present only in Extended OS status
4	Present only in Timing or Extended OS status
5	Linked only if there are heavyweight tasks in the system
6	Linked only if Resource is used by both tasks and ISRs
7	Linked only if Resource is used only by tasks
8	Linked only if Chaining task is Lightweight
9	Linked only if Chaining task is Heavyweight
10	Linked only if Idle task is the only extended task in the system
11	Linked only if calling Extended task uses floating-point
12	Linked only if neither Pre- nor Post-TaskHook is used
13	Linked only if Pre- or Post-TaskHook is used
14	Linked only if there are no flags, message queues, or message resources in the system, and COM status is not requested
15	Linked only if there are any flags, message queues, or message resources in the system, or COM status is requested
16	Linked only if `USEGETSERVICEID = FALSE` and `USEPARAMETERACCESS = FALSE`
17	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = FALSE`
18	Linked only if `USEGETSERVICEID = TRUE` and `USEPARAMETERACCESS = TRUE`
19	Linked only for basic, single-activation, lightweight, floating-point tasks
20	Linked only for basic, single-activation, heavyweight, integer tasks
21	Linked only for basic, single-activation, heavyweight, floating-point tasks
22	Linked only for basic, multiple-activation, lightweight, integer tasks
23	Linked only for basic, multiple-activation, lightweight, floating-point tasks
24	Linked only for basic, multiple-activation, heavyweight, integer tasks
25	Linked only for basic, multiple-activation, heavyweight, floating-point tasks
26	Linked only for extended, unique priority, integer tasks
27	Linked only for extended, unique priority, floating-point tasks
28	Linked only for extended, shared priority, integer tasks
29	Linked only for extended, shared priority, floating-point tasks
30	Implemented as a macro, so no code is linked
31	Not required on some targets

4.2.4 Reserved Hardware Resources

Timer units, interrupts, traps and other hardware resources are not reserved by SSX5.

4.3 Performance

The collection of performance data for the 16LX/FUJITSU port of SSX5 was achieved using a timer running four times slower than the CPU clock speed. The figures in this section, therefore, have an uncertainty level of up to four CPU cycles. The actual times are between zero and four cycles shorter than those reported in the remainder of this section.

Furthermore, the figures may be affected by the interaction of the target tools with measurement code running on the target hardware.

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	244	344	496	256	328	508
	NS	220	320	468	232	300	488
	KL	152	244	400	160	228	412
TerminateTask	LExt	0	0	0	0	0	0
	H	456	460	488	456	460	480
ChainTask	SWL	688	776	1020	884	936	1220
	SWH	872	972	1204	1068	1136	1396
	NSL	692	776	1020	884	940	1220
	NSH	864	968	1196	1060	1128	1388
Schedule	SW	212	212	248	208	212	240
GetTaskID		80	80	80	80	80	76
GetTaskState		252	248	252	284	280	280
EnableAllInterrupts		68	64	68	64	68	60
DisableAllInterrupts		60	64	60	60	60	56
ResumeAllInterrupts		124	128	128	128	128	124
SuspendAllInterrupts		76	76	76	80	76	76
ResumeOSInterrupts		128	124	128	128	128	124
SuspendOSInterrupts		76	76	80	76	80	72

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
GetResource	Task	104	108	116	108	108	112
	Combined	232	232	232	236	236	228
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	216	216	220	216	216	212
	Combined	232	236	236	232	232	228
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	264	268	292
	NS	n/a	n/a	n/a	268	264	284
	KL	n/a	n/a	n/a	200	200	220
ClearEvent		n/a	n/a	n/a	156	156	152
GetEvent		n/a	n/a	n/a	92	96	92
WaitEvent	<default>	n/a	n/a	n/a	1784	1792	1960
	fp	n/a	n/a	n/a	1808	1812	1980
GetAlarmBase		280	276	280	280	276	276
GetAlarm		276	276	280	276	280	272
SetRelAlarm		288	288	284	288	288	284
SetAbsAlarm		300	296	296	300	296	292
CancelAlarm		232	232	232	232	232	228
InitCounter		212	212	216	212	216	208
GetCounterValue		220	220	220	220	220	216
osek_tick_alarm	<default>	256	260	256	256	256	256
	KL	168	168	168	168	168	164
osek_incr_counter		32	32	28	32	32	28
GetActiveApplicationMode		16	16	16	16	12	12
StartOS		2112	2112	2112	2112	2112	2116
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	120	124	124	120	120	120
InitCOM		24	24	24	24	24	20
CloseCOM		24	24	28	28	28	20
StartCOM		116	112	116	452	452	440
StopCOM		48	48	48	48	48	44
ReadFlag		n/a	n/a	n/a	20	20	16
ResetFlag		n/a	n/a	n/a	24	20	20
ReceiveMessage		200	196	200	940	940	936
GetMessageResource		n/a	n/a	n/a	324	324	320
ReleaseMessageResource		n/a	n/a	n/a	380	376	372
GetMessageStatus		n/a	n/a	n/a	140	140	136
SendMessage	SW	464	560	708	1184	1256	1436
	NS	440	532	684	1160	1228	1412
	KL	296	388	540	1012	1080	1260

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
ActivateTaskset	SW	200	864	1080	224	868	1092
	NS	180	836	1056	196	840	1068
	KL	68	756	960	120	756	984
	SW2	204	864	1080	224	864	1088
	NS2	176	836	1052	196	840	1068
	KL2	68	752	960	120	756	984
ChainTaskset	SWL	664	1320	1616	852	1484	1800
	SWH	868	1500	1792	1056	1672	1988
	NSL	664	1320	1616	852	1484	1800
	NSH	860	1496	1784	1048	1664	1976
GetTasksetRef		80	76	80	80	76	76
MergeTaskset		200	200	200	200	200	196
AssignTaskset		80	76	76	80	76	72
RemoveTaskset		200	204	200	200	204	196
TestSubTaskset		252	252	252	252	252	248
TestEquivalentTaskset		232	236	232	232	236	228
TickSchedule	SW	376	1112	1316	476	1136	1364
	NS	352	1084	1292	452	1112	1336
	KL	280	1012	1220	380	1040	1264
	SW2	376	1112	1316	476	1112	1340
	NS2	352	1088	1292	452	1088	1312
	KL2	280	1016	1216	380	1016	1240
AdvanceSchedule	SW	332	1064	1272	432	1092	1316
	NS	308	1044	1248	408	1068	1296
	KL	228	964	1168	332	992	1216
	SW2	332	1064	1268	432	1068	1296
	NS2	304	1040	1244	408	1044	1268
	KL2	232	964	1168	328	964	1192
StartSchedule		284	280	280	280	284	280
StopSchedule		236	236	236	236	232	232
GetScheduleStatus		272	272	272	272	268	268
GetScheduleValue		244	244	244	244	240	240
GetScheduleNext		84	84	84	84	80	80
SetScheduleNext		124	124	124	124	124	120
GetArrivalpointDelay		84	84	84	84	84	80
SetArrivalpointDelay		80	80	80	80	80	76
GetArrivalpointTasksetRef		72	72	72	72	72	68
GetArrivalpointNext		84	84	84	84	84	80
SetArrivalpointNext		80	80	80	80	80	76
TestArrivalpointWritable		92	96	96	96	96	88

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
GetExecutionTime		24	24	24	24	24	20
GetLargestExecutionTime		76	72	72	76	76	68
ResetLargestExecutionTime		36	36	36	36	36	32
GetStackOffset		96	92	92	96	96	88

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	244	344	492	252	328	512
	NS	220	316	468	232	304	488
	KL	148	244	396	156	228	412
TerminateTask	LExt	0	0	0	0	0	0
	H	940	952	976	948	952	976
ChainTask	SWL	1280	1360	1620	1472	1524	1812
	SWH	1436	1532	1780	1628	1696	1964
	NSL	1280	1360	1620	1468	1524	1816
	NSH	1424	1528	1740	1620	1688	1956
Schedule	SW	204	208	244	208	208	244
GetTaskID		80	84	80	80	80	80
GetTaskState		248	252	252	284	280	284
EnableAllInterrupts		60	68	64	68	68	68
DisableAllInterrupts		56	60	64	60	60	60
ResumeAllInterrupts		124	124	128	124	128	124
SuspendAllInterrupts		76	76	76	76	76	76
ResumeOSInterrupts		124	128	124	128	128	128
SuspendOSInterrupts		72	76	76	76	80	76
GetResource	Task	100	108	120	108	104	116
	Combined	228	236	232	232	232	232
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
ReleaseResource	Task	212	216	216	216	216	216
	Combined	232	232	232	232	236	236
	CLEx	n/a	n/a	n/a	n/a	n/a	n/a
SetEvent	SW	n/a	n/a	n/a	264	268	296
	NS	n/a	n/a	n/a	264	264	292
	KL	n/a	n/a	n/a	200	200	220

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
ClearEvent		n/a	n/a	n/a	156	156	156
GetEvent		n/a	n/a	n/a	96	96	96
WaitEvent	<default>	n/a	n/a	n/a	2248	2284	2456
	fp	n/a	n/a	n/a	2272	2304	2480
GetAlarmBase		276	280	280	276	280	276
GetAlarm		272	276	280	280	276	276
SetRelAlarm		284	288	288	288	288	288
SetAbsAlarm		292	300	296	300	296	296
CancelAlarm		228	232	232	232	232	232
InitCounter		208	212	216	216	212	216
GetCounterValue		216	220	220	220	220	220
osek_tick_alarm	<default>	256	256	260	256	260	256
	KL	164	168	168	168	168	168
osek_incr_counter		28	32	28	28	32	32
GetActiveApplicationMode		12	16	16	16	16	16
StartOS		4824	4820	4824	4820	4824	4820
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	116	120	120	124	124	120
InitCOM		20	24	24	24	24	24
CloseCOM		24	24	24	24	24	24
StartCOM		112	116	116	444	444	452
StopCOM		44	48	48	48	48	48
ReadFlag		n/a	n/a	n/a	20	20	20
ResetFlag		n/a	n/a	n/a	24	20	24
ReceiveMessage		196	200	200	940	940	940
GetMessageResource		n/a	n/a	n/a	324	324	324
ReleaseMessageResource		n/a	n/a	n/a	376	376	380
GetMessageStatus		n/a	n/a	n/a	140	140	140
SendMessage	SW	460	560	712	1180	1252	1440
	NS	432	532	684	1160	1228	1416
	KL	292	388	540	1008	1084	1268
ActivateTaskset	SW	200	864	1080	224	864	1096
	NS	176	836	1052	200	836	1072
	KL	64	756	960	120	760	984
	SW2	200	864	1080	224	864	1096
	NS2	176	836	1052	200	836	1068
	KL2	64	752	956	120	756	988
ChainTaskset	SWL	1224	1904	2216	1408	2068	2396
	SWH	1432	2064	2368	1616	2228	2556
	NSL	1224	1904	2216	1408	2072	2396
	NSH	1424	2056	2356	1580	2220	2520

Configuration		**Application Uses**					
Events		**No**			**Yes**		
Shared Task Priorities		**No**		**Yes**	**No**		**Yes**
Multiple Task Activations		**No**	**Yes**		**No**	**Yes**	
GetTasksetRef		76	80	76	80	80	76
MergeTaskset		196	200	200	200	200	200
AssignTaskset		72	76	76	76	80	76
RemoveTaskset		196	200	204	200	200	204
TestSubTaskset		248	252	252	252	252	252
TestEquivalentTaskset		228	232	236	232	232	236
TickSchedule	SW	372	1112	1316	476	1136	1364
	NS	348	1088	1292	452	1112	1340
	KL	272	1012	1216	380	1040	1268
	SW2	372	1108	1312	476	1112	1340
	NS2	348	1084	1292	452	1092	1316
	KL2	276	1012	1220	380	1016	1248
AdvanceSchedule	SW	328	1068	1268	428	1092	1320
	NS	304	1040	1248	408	1068	1296
	KL	228	964	1168	328	988	1220
	SW2	328	1064	1272	432	1068	1296
	NS2	304	1040	1244	408	1044	1272
	KL2	224	964	1168	332	968	1196
StartSchedule		280	284	280	284	284	280
StopSchedule		232	236	236	236	236	236
GetScheduleStatus		268	272	272	272	272	272
GetScheduleValue		240	244	244	244	244	244
GetScheduleNext		76	80	84	80	80	84
SetScheduleNext		124	128	124	128	128	124
GetArrivalpointDelay		80	84	84	84	84	84
SetArrivalpointDelay		76	80	80	80	80	80
GetArrivalpointTasksetRef		68	72	72	72	72	72
GetArrivalpointNext		76	80	84	80	80	84
SetArrivalpointNext		76	80	80	80	80	80
TestArrivalpointWritable		92	96	96	96	96	96
GetExecutionTime		252	256	256	256	260	256
GetLargestExecutionTime		136	140	140	140	140	140
ResetLargestExecutionTime		116	120	120	120	120	120
GetStackOffset		88	92	92	92	96	92

Extended

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Service	Variant						
ActivateTask	SW	836	928	1076	840	912	1084
	NS	916	1012	1160	924	996	1176
	KL	688	776	936	700	764	948
TerminateTask	LExt	1020	1020	1044	1020	1020	1040
	H	1176	1172	1200	1176	1176	1196
ChainTask	SWL	2044	2092	2380	2228	2292	2564
	SWH	2188	2284	2540	2380	2444	2728
	NSL	2152	2212	2496	2332	2400	2688
	NSH	2296	2392	2648	2484	2548	2840
Schedule	SW	340	344	380	344	344	376
GetTaskID		96	92	92	92	92	92
GetTaskState		860	860	860	872	872	868
EnableAllInterrupts		76	76	76	76	76	76
DisableAllInterrupts		76	76	76	76	76	72
ResumeAllInterrupts		148	152	148	148	148	144
SuspendAllInterrupts		92	88	92	92	92	84
ResumeOSInterrupts		152	148	152	152	152	144
SuspendOSInterrupts		88	88	88	88	88	88
GetResource	Task	1232	1228	856	1336	1336	960
	Combined	792	788	788	896	892	892
	CLEx	864	860	864	968	968	960
ReleaseResource	Task	844	840	840	948	948	944
	Combined	784	788	788	892	888	884
	CLEx	812	816	812	920	920	916
SetEvent	SW	n/a	n/a	n/a	888	888	892
	NS	n/a	n/a	n/a	936	936	936
	KL	n/a	n/a	n/a	776	776	788
ClearEvent		n/a	n/a	n/a	252	248	244
GetEvent		n/a	n/a	n/a	676	676	676
WaitEvent	<default>	n/a	n/a	n/a	2496	2496	2644
	fp	n/a	n/a	n/a	2516	2520	2668
GetAlarmBase		712	712	712	712	716	712
GetAlarm		716	720	720	716	720	716
SetRelAlarm		828	824	828	824	828	824
SetAbsAlarm		800	800	800	800	800	796
CancelAlarm		672	672	672	672	672	668
InitCounter		664	668	664	664	664	664
GetCounterValue		636	636	636	636	636	632

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
osek_tick_alarm	<default>	336	336	332	332	336	332
	KL	168	168	168	168	168	164
osek_incr_counter		28	32	28	32	32	24
GetActiveApplicationMode		16	16	16	16	12	12
StartOS		5000	5000	5000	5000	5000	5000
ShutdownOS	NoHook	n/a	n/a	n/a	n/a	n/a	n/a
	Hook	132	128	128	128	132	128
InitCOM		24	24	24	24	24	20
CloseCOM		24	24	28	24	24	20
StartCOM		132	132	132	464	464	464
StopCOM		72	72	72	72	72	68
ReadFlag		n/a	n/a	n/a	84	84	80
ResetFlag		n/a	n/a	n/a	128	128	124
ReceiveMessage		564	564	564	1284	1284	1280
GetMessageResource		n/a	n/a	n/a	1348	1348	1340
ReleaseMessageResource		n/a	n/a	n/a	1340	1336	1336
GetMessageStatus		n/a	n/a	n/a	440	440	436
SendMessage	SW	1412	1504	1656	2116	2184	2356
	NS	1492	1592	1732	2196	2264	2444
	KL	1152	1244	1404	1860	1924	2108
ActivateTaskset	SW	1076	1776	1996	1096	1732	1984
	NS	1152	1848	2068	1168	1804	2052
	KL	932	1628	1848	948	1584	1832
	SW2	1080	1772	1996	1096	1732	1984
	NS2	1152	1848	2068	1164	1808	2056
	KL2	932	1628	1848	948	1584	1832
ChainTaskset	SWL	2340	3016	3352	2528	3168	3496
	SWH	2496	3204	3504	2684	3320	3648
	NSL	2440	3116	3456	2632	3264	3596
	NSH	2588	3296	3600	2780	3412	3740
GetTasksetRef		628	632	628	628	632	624
MergeTaskset		452	452	456	452	452	452
AssignTaskset		292	288	288	292	288	284
RemoveTaskset		452	452	452	452	452	448
TestSubTaskset		492	492	492	492	492	488
TestEquivalentTaskset		484	484	484	484	484	480
TickSchedule	SW	620	2120	2340	1440	2148	2404
	NS	724	2228	2448	1548	2260	2512
	KL	476	1976	2196	1292	2004	2256
	SW2	620	2120	2344	1440	2076	2328
	NS2	724	2232	2452	1552	2188	2436
	KL2	476	1976	2192	1296	1928	2176

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
AdvanceSchedule	SW	596	2092	2312	1412	2120	2372
	NS	700	2192	2408	1512	2216	2468
	KL	456	1948	2168	1268	1976	2228
	SW2	596	2092	2312	1412	2044	2296
	NS2	704	2192	2408	1512	2144	2396
	KL2	456	1952	2168	1268	1904	2152
StartSchedule		468	468	468	468	468	464
StopSchedule		372	372	368	372	368	368
GetScheduleStatus		404	400	404	400	404	400
GetScheduleValue		376	376	372	376	372	372
GetScheduleNext		180	180	180	180	180	176
SetScheduleNext		292	288	292	288	292	288
GetArrivalpointDelay		224	220	224	220	224	220
SetArrivalpointDelay		248	252	248	252	248	244
GetArrivalpointTasksetRef		188	188	192	188	192	184
GetArrivalpointNext		192	188	192	188	192	188
SetArrivalpointNext		320	316	320	316	320	316
TestArrivalpointWritable		212	208	208	208	208	208
GetExecutionTime		352	352	348	348	348	344
GetLargestExecutionTime		612	612	612	612	616	612
ResetLargestExecutionTime		588	588	592	592	592	588
GetStackOffset		92	92	96	96	92	92

4.3.2 OS Start-up Time

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

4.3.3 Interrupt Latencies

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

Standard

Configuration		Application Uses					
Events		No			Yes		
Shared Task Priorities		No		Yes	No		Yes
Multiple Task Activations		No	Yes		No	Yes	
Operation	ISR Category						
ISR Latency	Cat 1	52	52	52	52	52	52
	Cat 2	168	168	168	168	168	168

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	52	52	52	52	52	52
	Cat 2	388	388	388	388	388	388

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	52	52	52	52	52	52
	Cat 2	388	388	388	388	388	388

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.

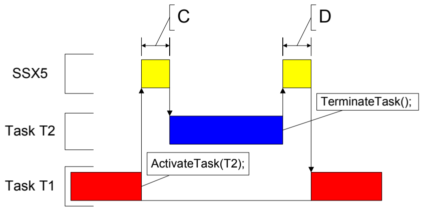

Figure 1: Task Activates a Higher Priority Task which Terminates Normally

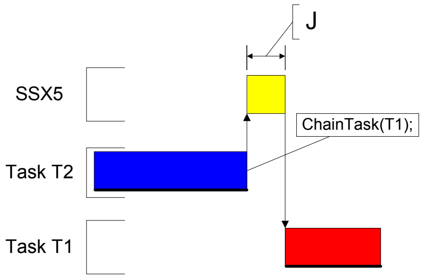

Figure 2: Task Chaining

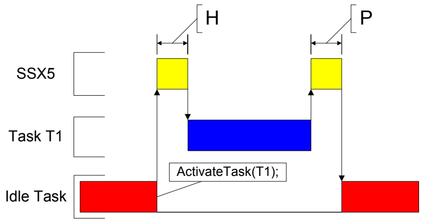

Figure 3: Task Activation from Idle Task

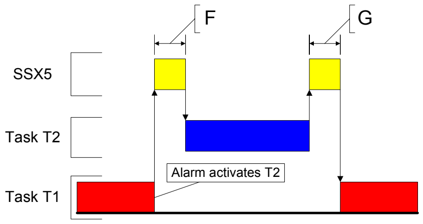

Figure 4: Task Activation from an Alarm

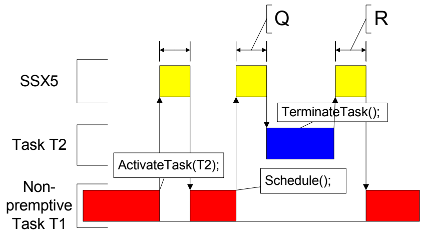

Figure 5: Non-Preemptive Task Calls Schedule()

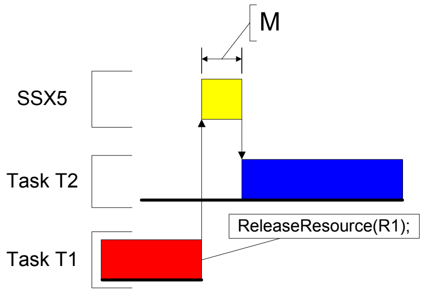

Figure 6: Blocked Task Activated by ReleaseResource()

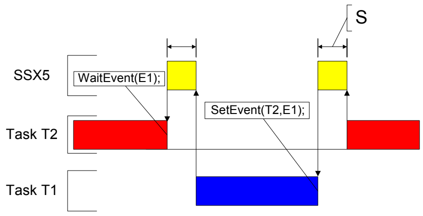

Figure 7: Waiting Task Activated by SetEvent()

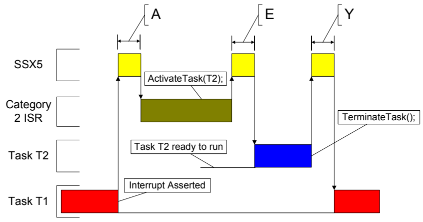

Figure 8: Category 2 ISR Activates a Higher Priority Task

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	196	300	436	196	300	432
Figure 1: D	Heavy, Basic/Extended	452	536	672	564	568	696
ChainTask	Light, Basic	484	628	880	492	628	916
Figure 2: J	Heavy, Basic/Extended	1156	1396	1768	1272	1432	1828
Pre-emption	Light, Basic	392	548	796	400	548	828
Figure 1: C	Heavy, Basic/Extended	640	740	980	836	904	1176
From idle task	Light, Basic	388	548	792	396	548	828
Figure 3: H	Heavy, Basic/Extended	640	736	980	836	900	1176
Triggered by alarm	Light, Basic	704	856	1108	712	860	1136
Figure 4: F	Heavy, Basic/Extended	948	1048	1288	1144	1212	1484
Schedule	Light, Basic	332	400	548	332	400	548
Figure 5: Q	Heavy, Basic/Extended	580	588	736	768	772	912
Release resource	Light, Basic	384	448	560	380	448	556
Figure 6: M	Heavy, Basic/Extended	632	636	748	816	820	920
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	1904	1904	2232
From category 2 ISR	Light, Basic	284	356	464	288	356	460
Figure 8: E	Heavy, Basic/Extended	536	544	652	724	724	828

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	708	788	928	712	788	932
Figure 1: D	Heavy, Basic/Extended	940	1000	1144	1028	1036	1168
ChainTask	Light, Basic	1100	1224	1480	1108	1220	1516
Figure 2: J	Heavy, Basic/Extended	2260	2460	2840	2328	2464	2868
Pre-emption	Light, Basic	708	848	1096	720	848	1136
Figure 1: C	Heavy, Basic/Extended	932	1028	1284	1124	1192	1476
From idle task	Light, Basic	708	844	1096	720	844	1132
Figure 3: H	Heavy, Basic/Extended	932	1028	1280	1124	1192	1472
Triggered by alarm	Light, Basic	1020	1160	1408	1032	1160	1448
Figure 4: F	Heavy, Basic/Extended	1248	1340	1596	1436	1500	1784
Schedule	Light, Basic	652	696	852	656	700	852
Figure 5: Q	Heavy, Basic/Extended	876	880	1036	1060	1060	1208

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	700	744	864	704	748	860
Figure 6: M	Heavy, Basic/Extended	924	928	1048	1108	1108	1220
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	2160	2160	2500
From category 2 ISR	Light, Basic	1088	1136	1252	1092	1136	1252
Figure 8: E	Heavy, Basic/Extended	1312	1316	1436	1496	1496	1608

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	1004	1072	1216	1000	1072	1212
Figure 1: D	Heavy, Basic/Extended	1172	1228	1364	1264	1260	1396
ChainTask	Light, Basic	1856	1976	2236	1864	1992	2268
Figure 2: J	Heavy, Basic/Extended	3240	3432	3824	3312	3436	3856
Pre-emption	Light, Basic	1284	1420	1672	1292	1420	1692
Figure 1: C	Heavy, Basic/Extended	1512	1608	1860	1700	1768	2040
From idle task	Light, Basic	1280	1416	1668	1292	1416	1692
Figure 3: H	Heavy, Basic/Extended	1512	1604	1856	1696	1764	2036
Triggered by alarm	Light, Basic	1672	1808	2056	1680	1808	2080
Figure 4: F	Heavy, Basic/Extended	1900	1996	2244	2084	2156	2424
Schedule	Light, Basic	776	816	972	776	816	968
Figure 5: Q	Heavy, Basic/Extended	1004	1004	1160	1184	1184	1332
Release resource	Light, Basic	1280	1324	1440	1384	1428	1540
Figure 6: M	Heavy, Basic/Extended	1508	1512	1628	1792	1792	1900
SetEvent							
Figure 7: S	Heavy, Extended	n/a	n/a	n/a	2768	2772	3120
From category 2 ISR	Light, Basic	1184	1228	1344	1184	1228	1340
Figure 8: E	Heavy, Basic/Extended	1412	1416	1532	1592	1592	1700

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	42	42	46	42	42	46
BCC1 lightweight, floating-point	50	50	54	50	50	54
BCC1 heavyweight, integer	72	72	76	72	72	76
BCC1 heavyweight, floating-point	72	72	76	72	72	76
BCC2 lightweight, integer	n/a	50	58	n/a	50	58
BCC2 lightweight, floating-point	n/a	50	58	n/a	50	58
BCC2 heavyweight, integer	n/a	72	80	n/a	72	80
BCC2 heavyweight, floating-point	n/a	72	80	n/a	72	80
ECC1 heavyweight, integer	n/a	n/a	n/a	86	86	90
ECC1 heavyweight, floating-point	n/a	n/a	n/a	86	86	90
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	94
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	94
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	46	46	46	46	46	46
BCC1 lightweight, floating-point	54	54	54	54	54	54
BCC1 heavyweight, integer	76	76	76	76	76	76
BCC1 heavyweight, floating-point	76	76	76	76	76	76
BCC2 lightweight, integer	n/a	54	58	n/a	54	58
BCC2 lightweight, floating-point	n/a	54	58	n/a	54	58
BCC2 heavyweight, integer	n/a	76	80	n/a	76	80
BCC2 heavyweight, floating-point	n/a	76	80	n/a	76	80
ECC1 heavyweight, integer	n/a	n/a	n/a	90	90	90
ECC1 heavyweight, floating-point	n/a	n/a	n/a	90	90	90
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	94
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	94

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	60	60	64	60	60	64
BCC1 lightweight, floating-point	64	64	68	64	64	68
BCC1 heavyweight, integer	88	88	92	88	88	92
BCC1 heavyweight, floating-point	88	88	92	88	88	92
BCC2 lightweight, integer	n/a	68	76	n/a	68	76
BCC2 lightweight, floating-point	n/a	68	76	n/a	68	76
BCC2 heavyweight, integer	n/a	90	98	n/a	90	98
BCC2 heavyweight, floating-point	n/a	90	98	n/a	90	98
ECC1 heavyweight, integer	n/a	n/a	n/a	100	100	104
ECC1 heavyweight, floating-point	n/a	n/a	n/a	100	100	104
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	108
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	108
Pre- and/or Post-Task hooks used						
Task type						
BCC1 lightweight, integer	62	62	64	62	62	64
BCC1 lightweight, floating-point	66	66	68	66	66	68
BCC1 heavyweight, integer	90	90	92	90	90	92
BCC1 heavyweight, floating-point	90	90	92	90	90	92
BCC2 lightweight, integer	n/a	70	76	n/a	70	76
BCC2 lightweight, floating-point	n/a	70	76	n/a	70	76
BCC2 heavyweight, integer	n/a	92	98	n/a	92	98
BCC2 heavyweight, floating-point	n/a	92	98	n/a	92	98
ECC1 heavyweight, integer	n/a	n/a	n/a	102	102	104
ECC1 heavyweight, floating-point	n/a	n/a	n/a	102	102	104
ECC2 heavyweight, integer	n/a	n/a	n/a	n/a	n/a	108
ECC2 heavyweight, floating-point	n/a	n/a	n/a	n/a	n/a	108

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		60	60	64	60	60	64
BCC1 lightweight, floating-point		64	64	68	64	64	68
BCC1 heavyweight, integer		88	88	92	88	88	92
BCC1 heavyweight, floating-point		88	88	92	88	88	92
BCC2 lightweight, integer		n/a	68	76	n/a	68	76
BCC2 lightweight, floating-point		n/a	68	76	n/a	68	76
BCC2 heavyweight, integer		n/a	90	98	n/a	90	98
BCC2 heavyweight, floating-point		n/a	90	98	n/a	90	98
ECC1 heavyweight, integer		n/a	n/a	n/a	100	100	104
ECC1 heavyweight, floating-point		n/a	n/a	n/a	100	100	104
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	108
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	108
Pre- and/or Post-Task hooks used							
Task type							
BCC1 lightweight, integer		62	62	64	62	62	64
BCC1 lightweight, floating-point		66	66	68	66	66	68
BCC1 heavyweight, integer		90	90	92	90	90	92
BCC1 heavyweight, floating-point		90	90	92	90	90	92
BCC2 lightweight, integer		n/a	70	76	n/a	70	76
BCC2 lightweight, floating-point		n/a	70	76	n/a	70	76
BCC2 heavyweight, integer		n/a	92	98	n/a	92	98
BCC2 heavyweight, floating-point		n/a	92	98	n/a	92	98
ECC1 heavyweight, integer		n/a	n/a	n/a	102	102	104
ECC1 heavyweight, floating-point		n/a	n/a	n/a	102	102	104
ECC2 heavyweight, integer		n/a	n/a	n/a	n/a	n/a	108
ECC2 heavyweight, floating-point		n/a	n/a	n/a	n/a	n/a	108

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