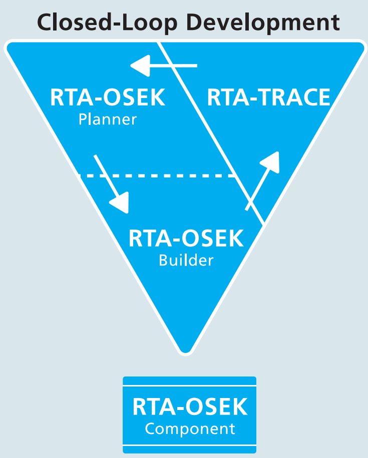
RTA-OSEK
Freescale S12X with the Cosmic Compiler



Features at a Glance

- OSEK/VDX OS v2.2 Certified OS
- RTOS overhead: 14 bytes RAM, 83 bytes ROM
- Category 2 interrupt latency: 61 CPU cycles
- Applications include: Body Electronics, HEVAC, Engine Management, Integrated Starter Alternators

ETAS Group
Automotive LifeCycle
Solutions

RTA-OSEK

RTA-OSEK provides an application design environment that combines the smallest and fastest OSEK RTOS with an unique timing analysis tool.

This data sheet discusses RTA-OSEK for the Freescale S12X and Cosmic compiler and should be read in conjunction with the Technical Product Overview "*Developing Embedded Real-Time Applications with RTA-OSEK*" available from ETAS.

The kernel element of RTA-OSEK is a fixed priority, pre-emptive real-time operating system that is compliant to the OSEK/VDX OS standard version 2.3 for all four conformance classes (BCC1, BCC2, ECC1 and ECC2) and intra processor communication using OSEK COM Conformance Classes A and B (CCCA and CCCB).

All CPU overheads of the kernel have low worst case bounds and little variability in execution time. The kernel is particularly suited to systems with very tight constraints on hardware costs and where run-time performance must be guaranteed.

The kernel is configured using an offline tool provided with RTA-OSEK. Determining in advance which features are used allows memory requirements to be minimized and API calls to be optimized for greatest efficiency.

All tasks and ISRs in RTA-OSEK run on a single stack – even extended tasks. This allows dramatic reductions in application stack space requirements.

The RTA-OSEK kernel is designed to be scalable. When a task uses queued activation or waits on events, the additional RTOS overhead required to support these features is paid by the task rather than by the system. This means that a basic single activation task uses the same resources in a BCC1 system as it does in an ECC2 system.

Compiler/Assembler/Linker

The libraries containing the code for the RTA-OSEK kernel have been built using the following tools:

- Cosmic cxs12x v4.7.10
- Cosmic cas12x v4.5.2

- Cosmic clnk v4.6.25

Memory Model

RTA-OSEK for the S12X supports the banked memory model provided by the Cosmic compiler. Kernel API calls use the *far* calling convention and the OS does not restrict their placement at link time. For runtime efficiency, and because the interrupt vector table only allows *near* function pointers, kernel interrupt wrappers and other internal kernel calls are *near* and must be placed in unbanked memory.

The functions are located in the linker section `os_unbank`, which should appear in the `seg` section of the linker file mapped to the fixed flash areas `0x4000-0x7FFF` or `0xC000-0xFF0F`. All API declarations and ISR calls are *far* and can be located in paged or unpaged memory. The CPU and compiler maintain the paging state implicitly.

In the banked memory model, it is not necessary to preserve the `PPAGE`, `EPAGE`, `GPAGE` and `RPAGE` registers during an ISR.

The kernel code expects to find its internal variables in *near* space. These variables are put in the `os_pir`, `os_pur`, `os_pid`, and `os_pird` sections, which the application must locate in unbanked memory.

ORTI Debugger Support

ORTI is the OSEK Run-Time Interface that is supported by RTA-OSEK for the following debuggers:

- HiWare HiWave
- iSYSTEM winIDEA
- Lauterbach TRACE32

Further information about ORTI for RTA-OSEK can be found in the *RTA-OSEK ORTI Guide*.

Hardware Environment

RTA-OSEK supports all variants of the Freescale S12X family: S12XA, S12XB, S12XD and S12XE.

Interrupt Model

RTA-OSEK supports the multilevel interrupt model through the 3-bit CCRH register and the legacy I bit in CCR. There are 7 regular interrupt priority levels above user level plus IPL 8, signified by the I bit being set. This is the level at which the non-maskable interrupts run.

Floating Point Support

RTA-OSEK is designed to work with the fully re-entrant software floating-point libraries supplied by Cosmic. This allows floating-point to be used in RTA-OSEK tasks and ISRs without the need to save and restore any additional context. `GPAGE`, `EPAGE` and `RPAGE` registers are saved in Floating point wrappers.

Evaluation Board Support

RTA-OSEK can be used with any Freescale S12X evaluation board. An example application is provided to run on the Softech SK-S12XDP512-A evaluation board. This application can be adapted for other target boards by adjusting the linker command file (to alter the RAM locations) and one source file (if alternative output pins are required).

Functionality

The table below outlines the restrictions on the maximum number of operating system objects allowed by RTA-OSEK.

	BCC1	BCC2	ECC1	ECC2
Max no of tasks	16 plus an idle task			
Max tasks per priority	1	16	1	16
Max queued activations	1	255	1	255
Max events per task	n/a	n/a	16	16
Max nested resources	255			
Max alarms	Not limited by RTA-OSEK			
Max standard resources	255			
Max internal resources	Not limited by RTA-OSEK			
Max application modes	255			

Note that OSEK specifies that queued activations in an ECC2 system are only possible for basic tasks. Where tasks share a priority level, the maximum number of queued activations per priority level is 255.

The number of alarms, tasksets, schedules and schedule arrivalpoints is only limited by available hardware resources.

Memory Usage

The memory overhead of RTA-OSEK is:

Memory Type	Overhead (bytes)
RAM	14
ROM/Flash	83

In addition to the RTOS overhead, each object used by an application has the following memory requirements:

Object	RAM Bytes	ROM Bytes
BCC1 task	0	23
BCC2 task	5	32
ECC1 task	11	35
ECC2 task	13	39
Category 1 ISR	0	0
Category 2 ISR	0	31
Resource	0	10
Internal Resource	0	0
Event	0	2
Alarm	9	31
Counter	4	82
ScheduleTable	9	86
ScheduleTable Expiry	0	10
Taskset (RW)	2	2
Taskset (RO)	0	2
Schedule	11	24
Arrivalpoint (RW)	8	8
Arrivalpoint (RO)	0	8

In addition to these static memory requirements each task priority and Category 2 interrupt has a stack overhead (in addition to application stack usage). The single stack model means that this overhead applies to each priority level rather than to each task. Similarly, for Category 2 interrupts this overhead applies for each unique interrupt priority. The table below shows stack usage for these objects.

Object	Stack Bytes
Task priority level	20
Category 2 interrupt	13

RTA-OSEK provides an optimization for task termination if the user can guarantee that tasks only terminate from their entry function. Tasks that terminate from elsewhere are not eligible for this optimization and duly require 8 more stack bytes per priority level than indicated in the table above.

Performance

The following table gives the key kernel timings for operating system behavior in CPU cycles.

Task Type	Basic	Extended	Ref
Category 1 ISR Latency	51	51	K
Category 2 ISR Entry Latency	61	61	A
Category 2 ISR Exit Latency	197	393	E
Normal Termination	131	317	D
ChainTask	297	787	J
Pre-emption	255	455	C
Triggered by alarm	517	717	F
Schedule	227	425	Q
ReleaseResource	259	461	M
SetEvent	n/a	815	S

All performance figures are for the non-optimized interface to RTA-OSEK. Using the optimized interface will result in shorter execution times for some operations. All tasks use lightweight termination and no pre or post task hooks were specified.

The execution time for every kernel API call is available on request from ETAS.

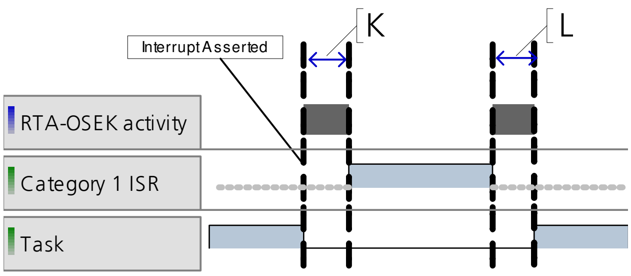


Figure 1 - Category 1 interrupt with return to interrupted task

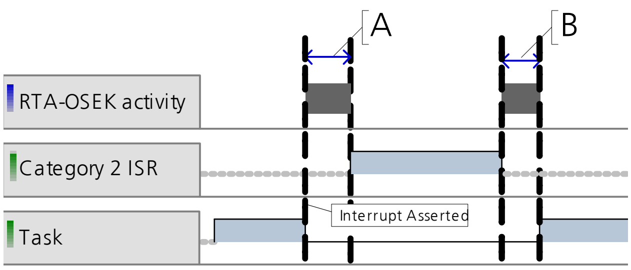


Figure 2 - Category 2 interrupt with return to interrupted task

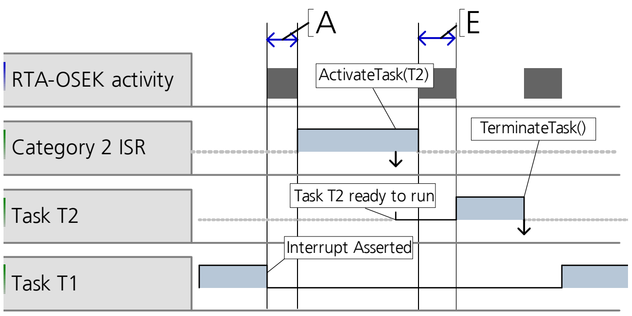


Figure 3 - Category 2 interrupt activates a higher priority task

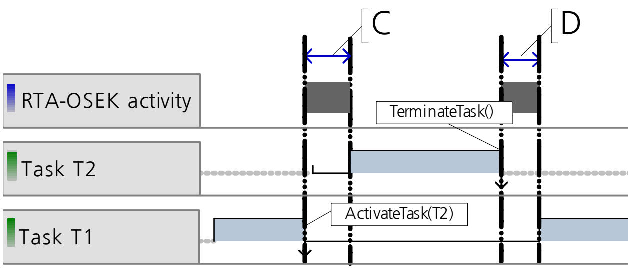


Figure 4 - Task activates a higher priority task

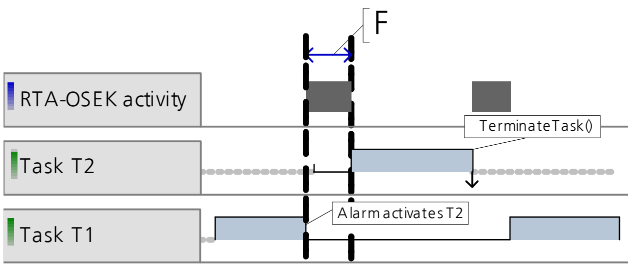


Figure 5 - Alarm activates task

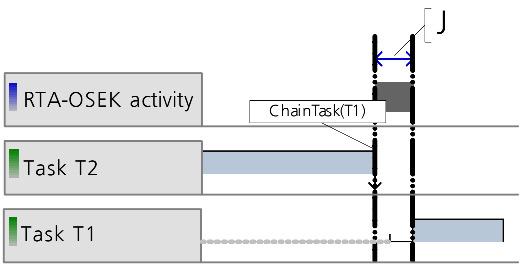


Figure 6 - Task chaining

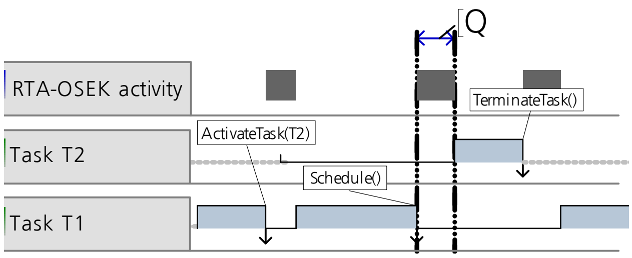


Figure 7 - Schedule() call

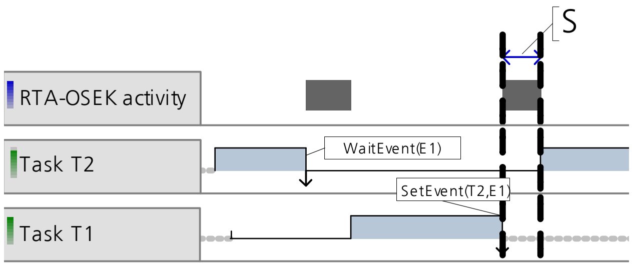


Figure 8 - Activation by SetEvent(

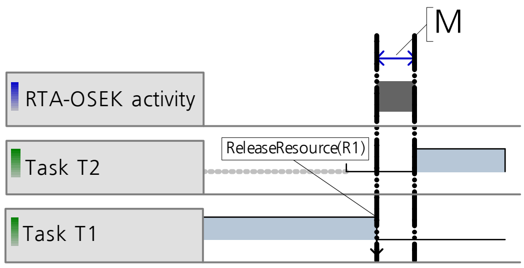


Figure 9 - ReleaseResource()

Benchmarks

The following sections shows benchmarks for RTA-OSEK memory usage for BCC1, BCC2, ECC1 and ECC2 conformant applications. The applications have the following framework:

- 8 tasks plus the idle task
- All basic tasks are lightweight tasks
- 1 Category 2 ISR with a 10ms minimum inter-arrival time
- 1 Counter
- 7 or 8 alarms, all attached to the same counter

- No resources or internal resources
- No hooks
- No schedules
- No tasksets
- Built using standard status

The following table shows the task priority configuration for each benchmark application:

Task/ISR	Stack (bytes)	Period (ms)	BCC1	BCC2	ECC1	ECC2
ISR1	10	10	IPL1	IPL1	IPL1	IPL1
A	10	10	8	8	8	8
B	20	20	7	7	7	7
C	30	20	6	6	6	6
D	40	30	5	5	5	5
E	50	50	4	4	4	4
F	60	80	3	3	3	3
G	70	100	2	2	2	2
H	80	150	1	1	1	2
Idle	10	-	idle	idle	idle	idle

The overhead figures give the ROM and RAM required for RTA-OSEK in addition to that required by the application. The RAM figure is shown split into RAM data and RAM stack.

BCC1

The BCC1 application uses 8 basic tasks with unique priorities. This application has the following overheads:

Memory Usage	Bytes
OS ROM	**1355**
OS RAM	**268**
comprising RAM data	103
comprising RAM stack	165

BCC2

The BCC2 application uses 8 basic tasks with unique priorities.

Tasks A-G are attached to 7 alarms. Task H is activated multiple times from Task A and has maximum queued activation count of 255.

This application has the following overheads:

Memory Usage	Bytes
OS ROM	**1541**
OS RAM	**264**
comprising RAM data	97
comprising RAM stack	167

ECC1

The ECC1 application uses 7 basic tasks and 1 extended task with unique priorities. Task H is the extended task and it waits on a single event that is set by basic tasks A-G.

This application has the following overheads:

Memory Usage	Bytes
OS ROM	**1871**
OS RAM	**288**
comprising RAM data	114
comprising RAM stack	174

ECC2

The ECC2 application uses 6 basic tasks and 2 extended tasks. Tasks G and H are the extended tasks and share a priority. The extended tasks wait on a single event that is set by tasks A-F.

This application has the following overheads:

Memory Usage	Bytes
OS ROM	**2350**
OS RAM	**340**
comprising RAM data	133
comprising RAM stack	207

Stack Optimization

Using stack optimization with the benchmark example identifies that the following tasks can share internal resources:

- Tasks A, B and C
- Tasks D, E and F
- Tasks G and H

The benefit of this optimization is shown in the following table:

Total Stack Space (bytes)	BCC1	BCC2	ECC1	ECC2
Non-optimized	**545**	**547**	**554**	**587**
OS Overhead	165	167	174	207
Application Overhead	380	380	380	380
Optimized	**250**	**250**	**259**	**259**
OS Overhead	70	70	79	79
Application Overhead	180	180	180	180

Notes

Contact Addresses

ETAS GmbH
70469 Stuttgart, Germany
Phone +49 711 89661-0
Fax +49 711 89661-106
sales.de@etas.com

ETAS S.A.S.
94588 Rungis Cedex, France
Phone +33 1 567000-50
Fax +33 1 567000-51
sales.fr@etas.com

ETAS Ltd.
Burton-upon-Trent
Staffordshire DE14 2WQ
Great Britain
Phone +44 1283 546512
Fax +44 1283 548767
sales.uk@etas.com

ETAS Inc.
Ann Arbor, MI 48103, USA
Phone +1 888 ETAS INC
Fax +1 734 997-9449
sales.us@etas.com

ETAS K.K.
Yokohama 220-6217, Japan
Phone +81 45 222-0900
Fax +81 45 222-0956
sales.jp@etas.com

ETAS Korea Co., Ltd.
Seoul 137-889, Korea
Phone +82 2 5747-016
Fax +82 2 5747-120
sales.kr@etas.com

ETAS (Shanghai) Co., Ltd.
Shanghai 200120, P.R. China
Phone +86 21 5037 2220
Fax +86 21 5037 2221
sales.cn@etas.com

ETAS Automotive India Pvt. Ltd.
Bangalore 560 068, India
Phone +91 80 4191 2585
Fax +91 80 4191 2586
sales.in@etas.com

www.etas.com

Subject to change (11/2008)

ETAS/COM-EU_Fi/02.2006