RTA-OSEK

STMicroelectronics ST30 with the ARM Compiler

Closed-Loop Development
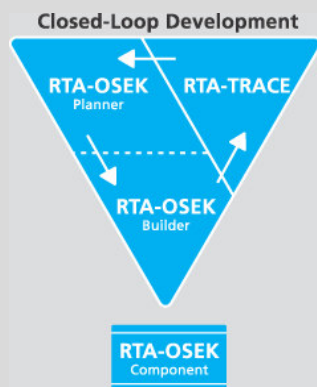
Features at a Glance

- OSEK/VDX OS version 2.2 certified OS
- RTOS overhead: 24 bytes RAM, 144 bytes ROM
- Category 2 interrupt latency: 118 CPU cycles
- Applications include: Car Body, Air Bag, Dashboard and Radio

RTA-OSEK

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

This port data sheet discusses the STMicroelectronics ST30 family port of the RTA-OSEK kernel alone and should be read in conjunction with the Technical Product Overview "*Developing Embedded Real-Time Applications with RTA-OSEK*" available from LiveDevices.

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 2.2 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:

- ARM Developer Suite V1.2

Memory Model

The ST30/ADS supports a flat 32-bit memory model. The only restrictions placed on memory usage are that locations used by on-board peripherals cannot be used for application code and the vector table must be located at 0x0.

ORTI Debugger Support

ORTI is the OSEK Run-Time Interface. Currently there are no ORTI

compatible debuggers supported by RTA-OSEK for this target.

Hardware Environment

RTA-OSEK supports all variants of the ST30 family including the ST30F771-Z, ST30F772-Z, ST30F771-A and ST30F774-Z.

Interrupt Model

RTA-OSEK for the ST30 architecture supports 17 interrupt priority levels. These correspond to the 16 levels of the ST30 Enhanced Interrupt Controller (EIC) and the 'F' bit (bit 6) of the current program status register (CPSR).

The ST30 architecture has an 8-entry vector table starting at 0x0. There are six processor exceptions, one reset vector and one reserved vector. The vector table can be provided either by the user or by RTA-OSEK. When multiple interrupt FIQ or IRQ interrupts are used in an application a 64-entry software vector table for the EIC exceptions is concatenated to the ST30 vector table.

OSEK Category 2 interrupts are only supported on the ST30 IRQ exception at EIC priority level 1, via the 64 EIC interrupt channels. When processing Category 2 interrupts, RTA-OSEK uses 16 bytes of the IRQ stack before reverting to the supervisor mode stack.

Floating Point Support

Tasks and ISRs may safely use software floating point.

Evaluation Board Support

RTA-OSEK for the ST30 can be used with any evaluation board. An example application is provided to run on the ST30F77x 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 below table outlines the restrictions on the maximum number of operating system objects allowed by RTA-OSEK.

	BCC1	BCC2	ECC1	ECC2
Max no of tasks	32 plus an idle task			
Max tasks per priority	1	32	1	32
Max queued activations	1	255	1	255
Max events per task	n/a	n/a	32	32
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 arrival-points is only limited by available hardware resources.

Memory Usage

The memory overhead of RTA-OSEK is:

Memory type	Overhead (bytes)
RAM	24
ROM/Flash	144

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	36
BCC2 task	10	52
ECC1 task	140	60
ECC2 task	142	68
Category 1 ISR	0	0
Category 2 ISR	0	52
Resource	0	20
Internal Resource	0	0
Event	0	4
Alarm	6	32
Counter	2	44
Taskset (RW)	4	4
Taskset (RO)	0	4
Schedule	12	36
Arrivalpoint (RW)	12	12
Arrivalpoint (RO)	0	12

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 below table shows stack usage for these objects.

Object	Stack Bytes
Task priority level	72
Category 2 interrupt	48

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 44 more stack bytes per priority level than indicated in the table above.

Performance

The following table gives the key kernel timings for operating sys-

tem behavior in CPU cycles.

Task Type	Basic	Extended	Ref
Category 1 ISR Latency	18	18	K
Category 2 ISR Latency	118	118	A
Normal Termination	211	393	D
ChainTask	313	739	J
Pre-emption	216	350	C
Triggered by alarm	445	579	F
Schedule	209	336	Q
ReleaseResource	217	344	M
SetEvent	n/a	578	S
Category 2 exit switch latency	112	239	E

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 LiveDevices.
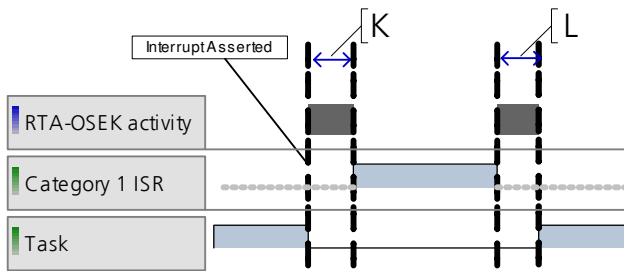
Figure 1 - Category 1 interrupt with return to interrupted task
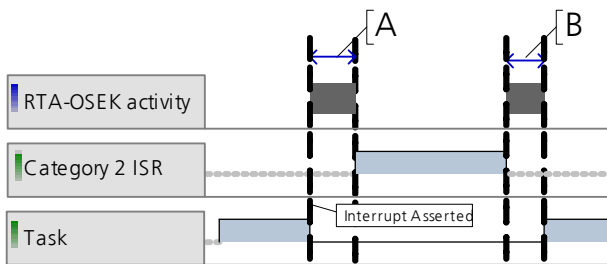
Figure 2 - Category 2 interrupt with return to interrupted task
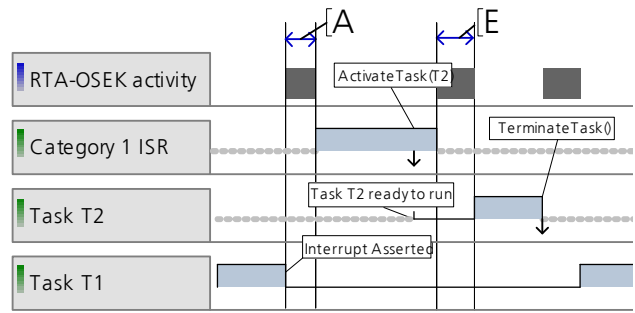
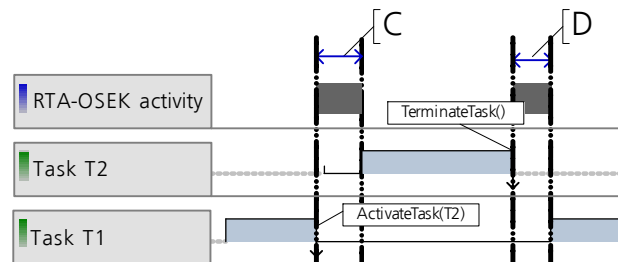Figure 3 - Category 2 interrupt activates a higher priority task

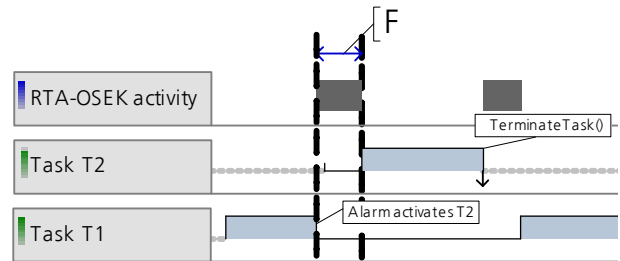Figure 4 - Task activates a higher priority task
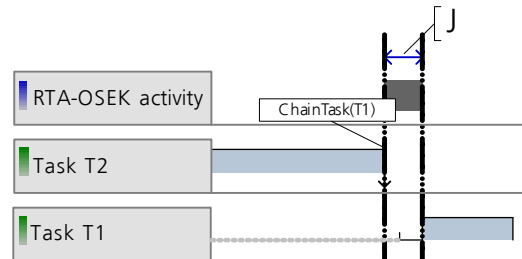

Figure 5 - Alarm activates task
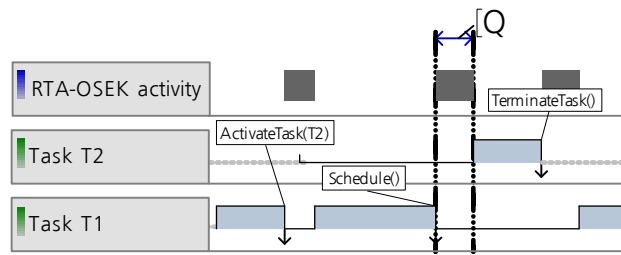

Figure 6 - Task chaining
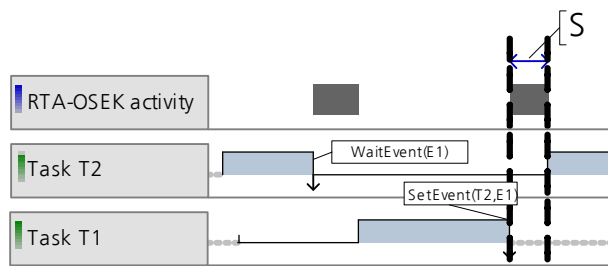
Figure 7 - Schedule() call
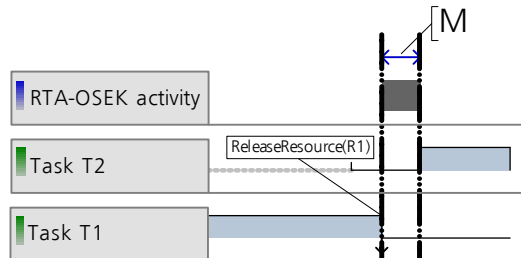

Figure 8 - Activation by SetEvent()

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	**1686**
OS RAM	**706**
comprising RAM data	74
comprising RAM stack	632

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	**1924**
OS RAM	**716**
comprising RAM data	76
comprising RAM stack	640

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	**2226**
OS RAM	**1022**
comprising RAM data	214
comprising RAM stack	808

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	**2632**
OS RAM	**1412**
comprising RAM data	364
comprising RAM stack	1048

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)	BCC 1	BCC 2	ECC 1	ECC 2
Non-optimized	**1012**	**1020**	**1188**	**1428**
OS Overhead	632	640	808	1048
Application Overhead	380	380	380	380
Optimized	**452**	**452**	**628**	**628**
OS Overhead	272	272	448	448
Application Overhead	180	180	180	180