RTA-OSEK

Xilinx PPC405 with the Wind River Compiler

Closed-Loop Development
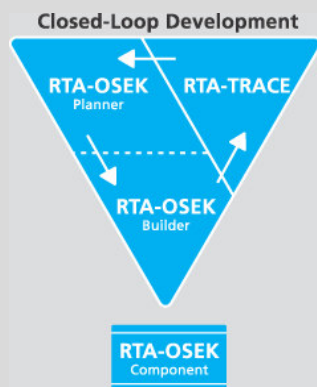
Features at a Glance

- OSEK/VDX OS version 2.2 certified OS
- RTOS overhead: 28 bytes RAM, 154 bytes ROM
- Category 2 interrupt latency: 1015 CPU cycles
- Applications include: Communications Gateways, Body Electronics

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 port of the RTA-OSEK kernel to the Xilinx PPC405 with the Wind River compiler 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:

- Wind River DCC v5.2.1.0
- Wind River DAS v5.2.1.0
- WInd River DLD v5.2.1.0

Memory Model

RTA-OSEK for the Xilinx PPC405 has a flat 32-bit memory model. For improved performance, RTA-OSEK uses a small amount of RAM data that should be located in the compiler's Small Data Area. For all other objects 32-bit addressing is used externally, thus placing no further restrictions on the location of user code

and data.

ORTI Debugger Support

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

• 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 Xilinx PPC405 family that use the OPB interrupt controller.

Interrupt Model

The RTA-OSEK port to the Xilinx PPC405 with the Wind River compiler supports 36 levels of interrupts. Category 1 and 2 interrupts handled by the OPB peripheral interrupt controller (INTC) can be configured with priorities in the range of 3 to 34. RTA-OSEK does not restrict the hardware location of the OPB interrupt controller. The CPU Periodic Timer Interrupt can be configured as a Category 2 interrupt with a priority of 1. The CPU Fixed Interval Timer interrupt can be configured as a Category 2 interrupt at priority 2 or a Category 1 interrupt at priority 35. All other CPU interrupts must be configured as Category 1 with a priority of 35.

Floating Point Support

The RTA-OSEK port to the Xilinx PPC405 with the Wind River compiler is designed to work with fully re-entrant software floating-point libraries supplied by Wind River Systems, Inc.. This allows floating-point to be used in RTA-OSEK tasks and ISRs without the need to save and restore any additional context.

Evaluation Board Support

The RTA-OSEK port to the Xilinx PPC405 with the Wind River compiler can be used with any evaluation board. An example application is provided to run on the Memic Design Virtex-II Pro 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.

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.

	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	255	255	255
Max internal resources	not limited by RTA-OSEK			
Max application modes	4294967295			

Memory Usage

The memory overhead of RTA-OSEK is:

Memory type	Overhead (bytes)
RAM	28
ROM/Flash	154

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	40
BCC2 task	10	56
ECC1 task	268	64
ECC2 task	270	72
Category 1 ISR	0	0
Category 2 ISR	0	52
Resource	0	20
Internal Resource	0	0
Event	0	4
Alarm	12	48
Counter	4	68
Taskset (RW)	4	4
Taskset (RO)	0	4
Schedule	16	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 table below shows stack usage for these objects.

Object	Stack Bytes
Task priority level	176
Category 2 interrupt	128

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 272 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 Entry Latency	484	484	K
Category 1 ISR Exit Latency	730	730	L
Category 2 ISR Entry Latency	1015	1033	A
Category 2 ISR Exit Latency	1477	1540	B
Normal Termination	1570	4210	D
ChainTask	2938	7513	J
Pre-emption	2863	4882	C
Triggered by alarm	4966	7048	F
Schedule	2698	4636	Q
ReleaseResource	2929	4876	M
SetEvent	n/a	8557	S
Category 2 exit switch latency	1987	3940	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.
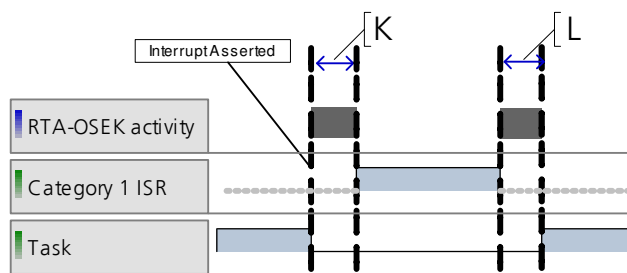
Figure 1 - Category 1 interrupt with return to interrupted task
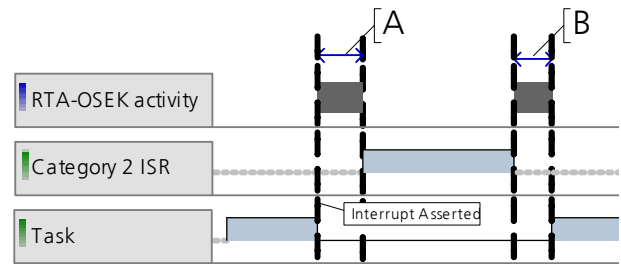
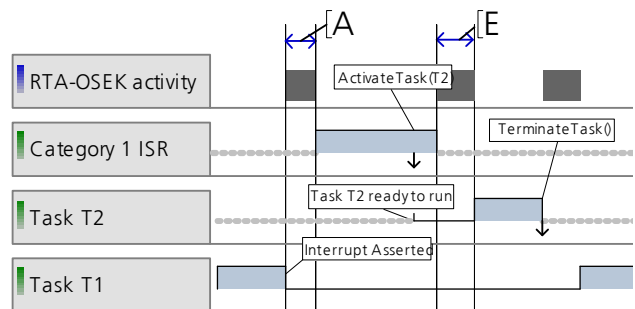Figure 2 - Category 2 interrupt with return to interrupted task

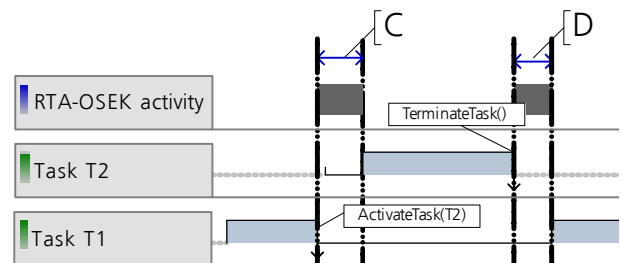Figure 3 - Category 2 interrupt activates a higher priority task

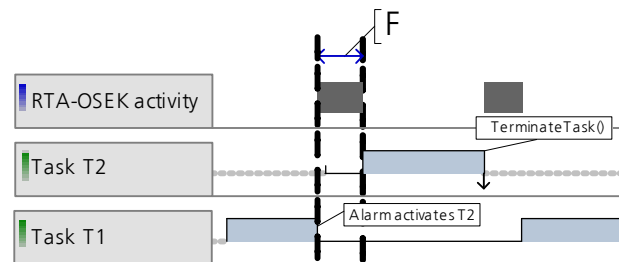Figure 4 - Task activates a higher priority task

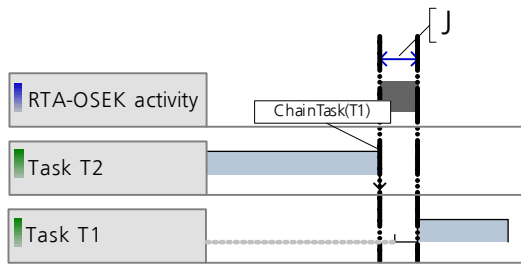Figure 5 - Alarm activates task

Figure 6 - Task chaining

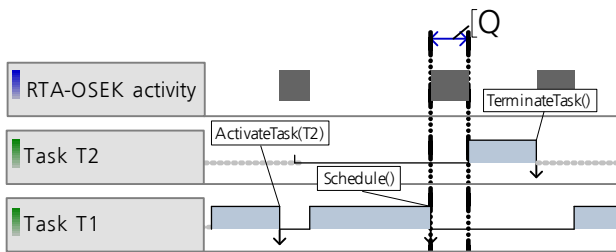
Figure 7 - Schedule() call

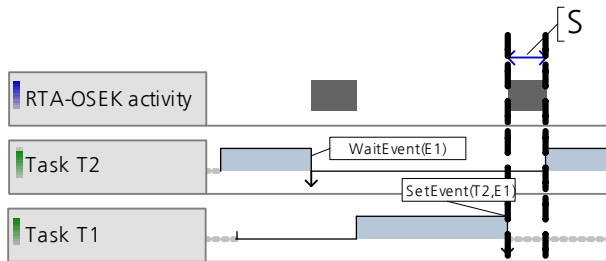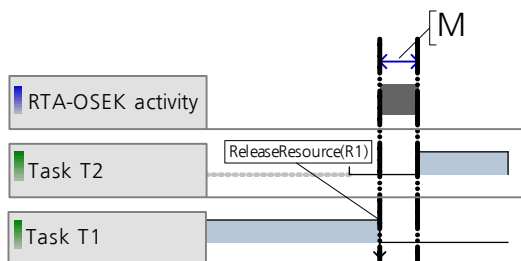
Figure 8 - Activation by SetEvent()

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	**2358**
OS RAM	**1656**
comprising RAM data	128
comprising RAM stack	1528

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	**2910**
OS RAM	**1668**
comprising RAM data	124
comprising RAM stack	1544

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	**3690**
OS RAM	**2212**
comprising RAM data	396
comprising RAM stack	1816

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	**4462**
OS RAM	**2906**
comprising RAM data	674
comprising RAM stack	2232

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	**1908**	**1924**	**2196**	**2612**
OS Overhead	1528	1544	1816	2232
Application Overhead	380	380	380	380
Optimized	**828**	**828**	**1116**	**1116**
OS Overhead	648	648	936	936
Application Overhead	180	180	180	180