Porting applications over the various conformance classes of Erika Enterprise

Quick guide

version: 1.0.1
December 11, 2012

EVIDENCE®
EMBEDDING TECHNOLOGY

About Evidence S.r.l.

Evidence is a spin-off company of the ReTiS Lab of the Scuola Superiore S. Anna, Pisa, Italy. We are experts in the domain of embedded and real-time systems with a deep knowledge of the design and specification of embedded SW. We keep providing significant advances in the state of the art of real-time analysis and multiprocessor scheduling. Our methodologies and tools aim at bringing innovative solutions for next-generation embedded systems architectures and designs, such as multiprocessor-on-a-chip, reconfigurable hardware, dynamic scheduling and much more!

Contact Info

Address:
Evidence Srl,
Via Carducci 56
Località Ghezzano
56010 S.Giuliano Terme
Pisa - Italy
Tel: +39 050 991 1122, +39 050 991 1224
Fax: +39 050 991 0812, +39 050 991 0855

For more information on Evidence Products, please send an e-mail to the following address: info@evidence.eu.com. Other informations about the Evidence product line can be found at the Evidence web site at: http://www.evidence.eu.com.

Contents

1 Introduction

The ERIKA Enterprise kernel provides various operating system APIs:

- a minimal multithreading API which offers multithreading and resource usage support for tiny microcontrollers, and

- a superset of the minimal API which follows more closely the OSEK/VDX OS specification.

Both APIs allow similar programming capabilities, being able to support multithreading applications for small microcontrollers.

However, there are a few differences which have to be taken into account when porting an application from the minimal API to the OSEK API and viceversa. The purpose of this document is to compare in detail the two APIs, for letting the user to choose the development platform more suitable to its needs.

2 Comparison between the various APIs

2.1 System differences

2.1.1 Conformance classes

Erika Enterprise supports four conformance classes named BCC1, BCC2, ECC1, and ECC2. The main idea is that these conformance classes contain a subset of the OSEK/VDX API features, allowing a fine tuning of performance vs code and memory footprint.

The minimal API supports a conformance class named FP, which is similar to the BCC2 conformance class of Erika Enterprise (or ECC2 if multistack is selected).

2.1.2 Error Handling

Erika Enterprise primitives typically return error values to inform the correct execution of the primitive. There are various error codes returned, which may be tuned to reduce the code footprint. In particular, there is support for an *extended status*, where the primitives return all the kind of errors which can be detected, and a *standard status*, where only part of the errors are raised.

To reduce the code footprint, the minilal API primitives typically do not return any error code. The system always assumes the correctness of parameter values, and acts in a default way upon particular conditions (e.g., task activations are dropped over a given number of pending activations). For this reason, moreover, there is no distinction between *extended status* and *standard status*.

Erika Enterprise supports the `ErrorHook` hook function and its macro, allowing the access to primitive parameters to cause the error. When defined in the OIL file, `ErrorHook` is called everytime an error different from `E_OK` is returned by a primitive. The minimal API does not support neither `ErrorHook` nor its macros.

2.1.3 PreTaskHook and PostTaskHook

Erika Enterprise supports the `PreTaskHook` and the `PostTaskHook` hook functions. These hooks are called by the kernel whenever a context change occurs. The minimal API does not support `PreTaskHook` and `PostTaskHook`.

2.1.4 System startup

Erika Enterprise supports system startup using `StartOS`. The application developer has to put a call to `StartOS` within the `main` function to start the system. In this way, the `main` function becomes the Background Task. A call to `StartOS` provoke the execution of the `StartupHook` hook function, the autostart of tasks and alarms if specified inside the OIL file. The `StartOS` primitive is also used to set the application mode.

The minimal API does not support `StartOS`, `StartupHook`, application modes and the autostart of tasks and alarms. With the minimal API, the kernel is already started at the first instruction of `main`. Even in this case, the `main` function is the Background Task. Task activations and Alarm arming at syystem startup must be done explicitly by the developer.

Warning: When porting an application from the minimal API to the OSEK/VDX API, a call to `StartOS` *must* be added in the system startup routine (typically `main`).

Warning: When using the minimal API, remember that activating a task within the `main` always causes a preemption to the activated task. This must be taken into account when more than one task has to be activated at startup. A possible solution is to activate the tasks starting from the highest priority one.

2.1.5 System Shutdown

In Erika Enterprise , the user should call `ShutdownOS` when the system must end. Calling `ShutdownOS` also causes a call to the `ShutdownHook` hook function.

The minimal API does not support `ShutdownOS` and `ShutdownHook`.

2.2 Tasks

2.2.1 Task termination

In Erika Enterprise, a task instance must always terminate with a call to `TerminateTask` or to `ChainTask`. Failing to terminate a task with one of these primitives brings to an undefined result; typically, it provokes an application crash. `TerminateTask` and `ChainTask` provide a simple way to clean and throw away the task stack.

The minimal API does not support `TerminateTask` and `ChainTask`. A task terminates at the last `}` of the task function. No explicit stack cleanup functions are supported.

Warning: When porting an application from the minimal API to the OSEK/VDX API, the developer *must* add a call to `TerminateTask` at the end of every task body.

2.2.2 Informations on tasks

Erika Enterprise supports the `GetTaskID` and `GetTaskState` primitives to get information about the running task ID and the task statuses.

The minimal API does not support neither `GetTaskID` nor `GetTaskState`.

2.2.3 Basic tasks and extended tasks

Erika Enterprise distinguishes between *Basic* Tasks and *Extended* Tasks. Basic tasks typically run on a shared stack, whereas extended tasks must run on a private stack. Extended tasks are tasks which use events and counting semaphores.

The minimal API does not have an explicit dinstinction between basic and extended tasks. The designer must take care to call counting semaphores and blocking primitives only within tasks with a private stack.

2.2.4 Number of pending activations

Considering the conformance classes BCC1 and ECC1 of Erika Enterprise, tasks can have only one pending activation. In conformance classes BCC2 and ECC2, tasks can have more than one pending activation. The maximum number of pending activations is specified inside the OIL file and can not be changed at runtime. Pending activations of tasks with the same priority are processed in a FIFO order, meaning that the ready queue enqueues *activations* and not *tasks*, consuming RAM space for each pending activation which have to be stored.

When using the minimal API, tasks store the number of pending activations as an integer value. Therefore, the maximum value is implementation dependent. The developer can not rely on a particular order in the processing of pending activations of tasks with the same priority.

2.3 Interrupt handling

There is always a distinction between ISR type 1 and type 2.

While Erika Enterprise supports the primitives for disabling interrupts, the minimal API does not (please refer to the architecture manual for functions to disable interrupts).

2.4 Event handling

Erika Enterprise supports events for the two conformance classes: ECC1 and ECC2. Events are not supported by the minimal API.

2.5 Support for non-blocking semaphores

Erika Enterprise supports the non-blocking counting semaphores primitives in the BCC1 and BCC2 conformance classes. BCC1 and BCC2 can also run on a monostack configuration.

Semaphore primitives are supported by the minimal API only in the multistack configuration.

2.6 ORTI support

If configured, Erika Enterprise maintains information to support ORTI debugger awareness. The RT-Druid code generator is able to generate appropriate ORTI files which can be interpreted by debuggers such as Lauterbach Trace32.

The minimal API does not support ORTI kernel awareness.

3 History

Version	Comment
1.0.0	Initial version of this document.
1.0.1	Added new versioning mechanism.