

OSEK: a super-small kernel for deeply embedded applications?

The OSEK/VDX specification was developed by a consortium of automotive companies and suppliers, and has therefore sometimes been labeled as only being applicable to automotive applications. Although this label has been placed on OSEK, it is important to realize that the specification was designed to be applicable to a wide range of in-vehicle applications, and to efficiently target different word-length processors. In fact there are commercially available OSEK/VDX compliant operating systems available for a variety of 8, 16 and 32 bit microprocessors. The questions that will be addressed in this article are whether the specification is valid in other industries, and if so, for what types of applications? In order to answer these questions we need to restate the goals and the problems the specification is designed to solve.

THE OSEK/VDX SPECIFICATION

The OSEK/VDX operating system specification actually includes specifications for three different components, a kernel, a communications module, and a network management module. Each of these will be described briefly below.

There were several goals in the definition of the OSEK/VDX kernel, including isolation of the developers from the unnecessary details of their target hardware, supplying developers a rich set of kernel features and objects to simplify the implementation of embedded applications, and facilitating the integration of software developed by different entities. An entity is defined as another developer, team or supplier. The kernel specification defines a static configuration approach for scalability, a highly efficient scheduling policy, and the ability to be ROMable. The operating system specification was also designed to address stringent real-time requirements, minimal resource usage, reliability and cost sensitivity.

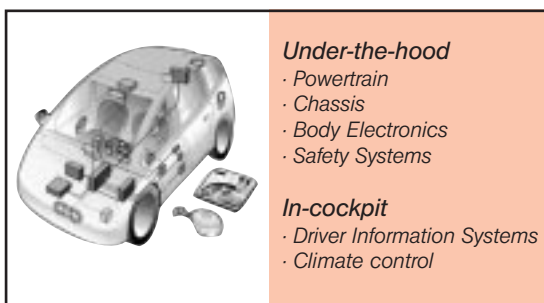


Figure 1. Automotive embedded systems.

The communication specification is designed to ensure the portability of application software and the independence from the underlying network hardware and communication protocols. It does this by defining a communication interface protocol for both the inter-ECU communication and the internal ECU (inter-task) communication.

The third part of the OSEK specification deals with network management. This specification focuses on in-vehicle network node monitoring for diagnostic support, start-up behavior of the network, reading and set-

ting of network- and node-specific parameters, etc.

USING OSEK FOR NON-AUTOMOTIVE APPLICATIONS

A quick glimpse at the goals of the specification clearly shows that the operating system kernel and the communication specifications have a common requirements base and goals that can be found in other industries. On the other hand, the network management specification is specifically targeted to automotive application needs. Although the label of "network management" could be related to the needs of other industries, the design of the specification does not generally apply to other industries and can even create a misunderstanding on the meaning of Network for users in other industries.

It is useful to divide the communication layer into two main domains. The first one is the inter-processor communication, which is independent of the underlying protocol (Typically CAN). The second one is the inter-task communication, which includes message queues, among other constructs. The CAN communication layer is mainly used in automotive applications, however CAN is also found in other markets, such as industrial automation.

The main area of interest in evaluating the usability of OSEK with other industries and applications is the operating system kernel. The OSEK operating system kernel specification is focused towards specifying a super-small scaleable kernel. With the continuous trend towards cutting costs to reduce the final product price, a scalable operating system with a minimum memory footprint is a common requirement for deeply embedded applications. Examples of such a requirement can be found in the storage application market (disk drives, DAT tapes, CD-ROM's...). In general, such applications require operating systems with a footprint below 5 kilobytes. The scalability of an OSEK compliant operating system via static configuration allows the designer to include only the required services for his particular application, thus minimizing the memory footprint of the kernel. Implementations of the OSEK specification, such as the pOSEK real-time operating system (RTOS) from Integrated Systems, range from

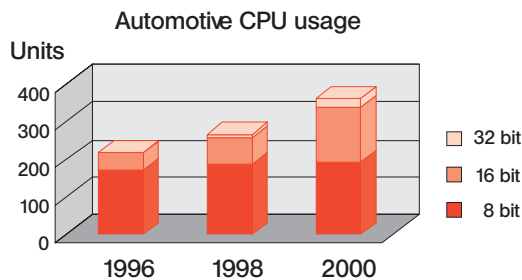


Figure 2. Automotive CPU explosion.

between 1 and 7 kilobytes, depending on the processor and the required operating system services. In addition, although some non-OSEK operating systems are statically configured, they do not offer a standardized interface, which allows application software to be easily ported from one OSEK implementation to another one or from one target processor to another one. Similar requirements for minimal size and maximum portability can also be found in home appliance applications where embedded software for refrigerators, dishwashers, microwave ovens and other appliances has a strict requirement for a minimal memory footprint.

There are other advantages to the fact that an OSEK operating system is a static kernel. For example, a major shortcoming of testing safety critical systems with dynamically allocated memory is that no complete test suite can be created to cover every possible combination of events and inputs that could create a potential error. Since an OSEK kernel is statically defined, all kernel objects are known at compilation time and dynamic memory allocation at run-time is not allowed. As a result, the amount of testing needed to ensure the correct behavior of the application is greatly reduced. This type of benefit is important for applications requiring any sort of certification such as medical instrumentation. In addition, these applications can benefit from the interrupt processing provided by OSEK operating systems. Developers need to design their software to spend as little time as possible at the interrupt level. OSEK operating systems offer a mechanism to efficiently set an event or activate a task from within an interrupt handler and then perform the requested call in non-interrupt mode.

OSEK also provides a number of standardized hook routines, which are not found in many traditional RTOS's. Hook routines have two main purposes: to be

	PowerPC 555	Infineon 167
Minimum	2 kBytes	1.5 kBytes
Maximum	7.5 kBytes	4.5 kBytes

Figure 4. pOSEK Kernel Size.

used in production application or to be used for debugging purposes. A hook such as ShutdownOS enables the designer to perform a clean shut down of the OS, taking into account any specific needs of his application or environment. This feature can be beneficial for many applications, such as those in the industrial automation field, where a robotic arm for example might need to be brought back to an initial position prior to shutting down the OS.

Priority inversion is a well-known problem for embedded system designers. The Mars pathfinder had a rather typical example of a priority inversion problem causing a total system reset. OSEK provides a simple mechanism to avoid such problems. The priority ceiling protocol, as specified in OSEK, ensures that tasks are only transferred from the ready state into the running state if all resources potentially occupied by that task during its execution have been released. As a result, avoiding priority inversion is solved intrinsically by the standardized management of resources (semaphores) using the priority ceiling protocol in the OSEK specification.

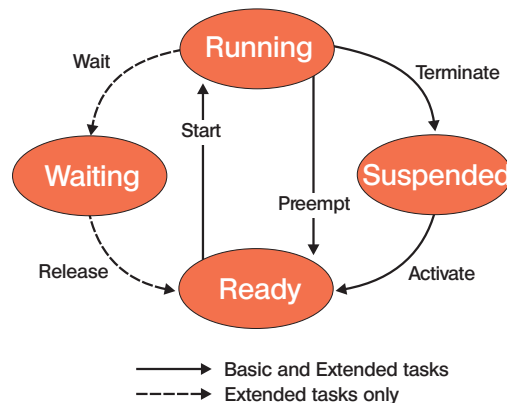


Figure 3. pOSEK Task State Model.

CHOOSING OSEK AS AN OPERATING SYSTEM

We have seen that much of the OSEK specified functionality can be used with various applications in many industries, including industrial automation, medical instrumentation, storage devices or aerospace applications. In addition, other industries such as telecom and consumer electronics products can benefit from the described features of OSEK as well.

While considering using an OSEK operating system such as pOSEK for any application, a developer should focus on his applications needs. Will a static kernel meet his requirements? Are the configurability of OSEK and a small memory footprint important aspects of the decision process? Is there a need to adhere to a standard API in order to easily transition from one processor to another one? Would he benefit from simplified testing? What types of services will be needed for this application, both now and in the future?

Many technical aspects have to be considered in such a decision. In addition, as OSEK provides a standard-

ized specification, another critical aspect to consider in selecting a specific OSEK operating system is the supplier itself: What type of support (technical and services) can the supplier provide worldwide? If necessary, can this supplier provide a complete solution including development tools (compilers, debuggers), services and higher graphical specification and code generation tools? What other expertise can the supplier offer?

OSEK represents an important step forward in providing a standardized RTOS specification for the automotive industry. But since automotive applications have the same types of constraints as other deeply embedded applications (cost sensitivity, application memory footprint, etc), the OSEK kernel and communication

specification have the potential to be used as a super-small operating system for deeply embedded applications elsewhere ■

Marc Serughetti is the Design Automation Solutions Marketing Manager at Integrated Systems, Sunnyvale, CA. Within Integrated Systems, he held positions with the European Sales Operation and the DAS Marketing Group.

Prior to joining Integrated Systems, Mr. Serughetti was with ABB Power Generation in Switzerland. He holds an Engineering Degree from the National Superior School of Aeronautical Engineering (ENSI-CA, France) and a Master of Science from the University of Washington.

It's hard to compete without the right tools.



NEW!
Celeron™ & Pentium III
processor
In-Circuit - Emulation

Since 1991, American Arium has worked closely with Intel to create powerful new development tools for each new Intel processor as it was introduced. So whether you're using Intel's Pentium® - Pentium pro or Pentium II processor, there's an American Arium tool designed to get your project to market ahead of schedule and under budget.

Pentium is a registered trademark of Intel Corporation.
Celeron is a trademark of Intel Corporation.



www.arium.com
E-mail: info@arium.com

P.O. Box 704
890 AD Purwijnen
The Netherlands

Tel: +31 77 207 84 08
Fax: +31 77 207 84 30
e-mail: info@logic.nl

www.logic.nl



There's always a Logic solution