Towards portable embedded automotive applications

by Dr. Jochen Schoof, 3SOFT

Combining OSEK/VDX and a hardware abstraction layer has finally taken the developers where OSEK/VDX alone promised to take them but did not succeed. Using this package will be the normal approach in future.

The combination of an OSEK/VDX conforming operating system and a widely accepted a hardware abstraction layer (HAL) makes it possible to write hardware-independent software. Hardware-independent software has a number of benefits. It decreases the costs of switching between architectures during development. It facilitates the re-use of applications or parts of them. And finally it makes it possible to easily run software in a simulated environment. OSEK/VDX simulators running under Windows are still available. Adding a HAL to simulate all required peripherals on the PC makes it possible to run and test the application at the developer's workplace. Combining OSEK/VDX and a HAL has finally taken the developers where OSEK/VDX alone promised to take them but did not succeed. Using this package will be the normal approach in future, just as using OSEK/VDX already is today.

OSEK/VDX characteristics

The OSEK/VDX-OS standard defines a static, scalable, multi-tasking real-time operating system. Static means the exact number of required resources has to be determined before generating a specific kernel. Dynamic changes at runtime are not supported. By setting up these restrictive demands the chances of better code optimization are increased significantly, making it possible to use even simple processors. Thus it is not necessary to estimate the resources required in the worst case, instead the actual requirements of the static system can be easily determined and the system adapted to these.

The most important resources of OSEK-OS are tasks, events and alarms. For working with these an API is provided specially designed for small size. An API conforming with the current version 2.2 of the standard has to provide about 40 user functions. The API's small size enables the developer to learn and use it easily.

Tasks can be compared to ordinary C functions. The only difference is that tasks can be pre-empted at any time. OSEK/VDX demands that a pre-empted task will later be resumed at the very point of pre-emption. From inside the task the pre-emption is invisible. The idea behind pre-emption of tasks is to provide means to reach more dynamic applications. They no longer have a linear, continuously repeating flow of operation but instead can react to a given situation more quickly. This approach is called event-driven programming.

Two kinds of tasks are provided in OSEK-OS: basic and extended. Basic tasks only release the processor by terminating themselves and cannot wait for any event. Therefore they typically perform jobs, which have to be completed entirely after activation. Extended tasks however can wait for events, so they usually only need to be started once and are controlled by appropriate events. Each task is given a fixed priority that must not be changed at run-time. This priority is used to determine which task will be performed. The scheduler simply selects the highest priority task that is ready to run.

Multi-tasking is the central concept of OSEK/VDX. All other services primarily help to facilitate working with pre-emptable, concurrent tasks. The application is formed by several tasks which in some way need to coordinate their activities. They can be synchronized by means of events or resources. Tasks can wait for events as well as cause them. Preconditions can thus be signaled to other tasks. This requires the participation of an extended task, because only these are able to wait for events. Basic tasks cannot do this, but they can cause events.

Resources in OSEK/VDX terminology refer to data structures similar to semaphores, which can be used to guarantee exclusive access to global data. This makes it possible to use them for deadlock prevention. OSEK/VDX uses the priority ceiling protocol for this purpose. It assigns a special priority to each resource and sets a task priority to this value while it uses the matching resource. This is a safe means of synchronization, because tasks using resources can neither wait for events nor be terminated. Additionally, the priority ceiling protocol is mathematically proven to be deadlock-free.

OSEK/VDX-OS also offers messages between tasks. Their behavior is defined in OSEK/VDX-COM, which also describes messages between ECUs. The main purpose of messages is to send data between tasks. But OSEK/VDX offers some interesting additional services. By sending

a message it is possible to automatically perform another activity. This can be activating a task, setting an event, executing a call-back function or setting a flag. So it is possible to activate the receiver of the message by just sending the message. This makes sure there always is a receiver for it.

Time management in OSEK/VDX uses counters and alarms. The unit of time is a tick of fixed length. An alarm is an activity to be performed when a counter reaches a given value. It results in activating a task, causing an event or executing a call-back routine. Alarms can be both activated and deactivated dynamically. It is possible to specify whether an alarm is to be performed once or cyclically. Although designed to measure time, counters can also be used to control different things. An example might be to have a counter count errors detected in an application. Every three errors an alarm operates and activates a task that stores the error codes in some given location.

Finally interrupt service routines (ISRs) are still available as expected by the programmer. They are divided into two categories. Category 1 ISRs execute without any interference with the operating system. They behave exactly the same as usual on the particular architecture used. As a drawback these routines cannot use any operating system services. Coordination with tasks has to be done using global variables or similar. In category 2 ISRs, on the other hand, the majority of operating system functions can be used. To achieve this a few instructions need to be inserted at the beginning and end of the ISR. These ISRs can now activate tasks to let them perform the less time-critical parts of the work. On architectures without interrupt priorities or levels this helps to receive the next interrupt request faster.

The services provided are completed by hook routines. These need to be written by the programmer, but are executed by the operating system in certain situations. The StartupHook for example is performed on start-up of the application and can be used to initialize on-chip peripherals. The second interesting hook is the ErrorHook which executes every time an OSEK/VDX API function returns an error value indicating some internal error. The ErrorHook is similar to a small exception handler.

ProOSEK features an integrated graphical configuration environment

Benefits and challenges

OSEK/VDX defines all the services a real-time multi-tasking kernel needs. It facilitates the design and implementation of software that can be ported to different architectures more easily. An application can for example be split into various functional parts. Each of these is implemented as one or more tasks. Different developers can work on different tasks and can even test them independently. This is possible because a task gets no information about the origin of activations or events sent to it. Thus to test a single task the programmer can write one more task that generates all the events and activations the first task needs to react to.

Also, the maintainability of applications is improved. First, with a standardized software underlying the application it is much easier to find someone familiar with it than it is for proprietary solutions. Second, functions that belong together can be put into one task to form a ready-to-use tested module. Typical change requests should now only influence a few tasks and so the changes only need to be performed locally. This can of course also be achieved without using OSEK/VDX. But the standard provides easy–to-use means which facilitate this approach.

However, the standard does not define everything that is needed to develop hardware independent software. OSEK/VDX-OS takes a few steps in this direction but leaves a few blank spots. It is possible to encapsulate all direct accesses to hardware into tasks. These tasks still require changes when programming for different hardware, but the changes only need to be applied locally. This approach has been used for some years now and programmers have begun to call for a better solution that makes applications portable without any changes. The key to reaching this goal is the definition of a standard driver interface. With the publication of this interface anybody would be able to write drivers for all kinds of peripherals. The driver functions could be used via a standardized interface that makes switching between different drivers easy. The software written on top of these drivers does not require any changes. The peripherals to be accessed can easily be determined via a special configuration tool quite similar to the configurators provided with OSEK/VDX implementations.

The solution: a hardware abstraction layer

The idea of a hardware abstraction layer (HAL) is simple: it offers one interface to the application and another one to the hardware driver. The HAL has responsibility for mapping requests from the application to the corresponding driver and vice versa. The mapping

OSEK/VDX is a standard for…

■ …modular and static real-time operating systems, initiated by leading automotive manufacturers and suppliers, research institutes and software developers. OSEK stands for the German words meaning open systems and the corresponding interfaces for automotive electronics". VDX means "vehicle distributed executive" and originally formed a separate initiative, which later joined the OSEK group. Both titles make the intention of OSEK/VDX clear: standardized interfaces for automotive electronics. The standard formulated is completely platform-independent and makes only relatively small demands on target systems. This allows use of rather simple processors in control tasks instead of more expensive solutions without added functionality. However, there is practically no upper bound for possible target platforms, because it is reasonable to use even more complex CPUs.

The standard defines several independent components: the actual operating system (OS), a communication subsystem (COM), a network management system (NM), the OSEK implementation language (OIL) and the OSEK run-time interface (ORTI). Two new standards have been added in recent years. They aim specially at the future market for x-by-wire systems providing time-triggered services. They are OSEK/VDX time-triggered operating system (OSEKtime) and fault-tolerant communication system (FTCOM). Currently the OSEK/VDX standards are being prepared for acceptance as ISO 17536. Today the operating system standard in particular is a mature document reflecting years of practical experience with conforming implementations. The first ECUs with OSEK-based software have been on the roads since 2000. The first car using OSEK in all its ECUs can already be bought. More and more automotive OEMs demand that software for their cars should use OSEK. ■

can be defined in a simple configuration language similar to OIL. Of course configuration tools can be provided to make writing these configuration files easier. All this is well known from OSEK/VDX.

The application now no longer has to access any hardware directly. Instead it calls functions provided either by the OSEK/VDX implementation or by the HAL. The HAL itself is again a hardware-independent piece of software using hardware specific drivers. So instead of porting an application, only the operating system – if not already available for the architecture anyway – and the drivers need to be ported. In contrast to porting the operating system, writing a driver is quite simple. The HAL driver interface is published so everyone can write a driver.

The hardest job when defining a HAL is to find supporters for the particular solution. A HAL will only help software developers if it is widely accepted. Unfortunately the chances of establishing another OSEK/VDX standard were quite bad. So the German car manufacturers Audi, BMW, DaimlerChrysler, Porsche and Volkswagen formed an interest group called HIS (Hersteller-Initiative Software i.e. manufacturer's software initiative), aiming to agree on standard approaches wherever possible. One of the first results was the definition of a HAL called I/O-Library.

This solution has been defined and implemented in a joint effort between DaimlerChrysler and 3SOFT, the latter being well known as the developer of the ProOSEK operating system. 3SOFT has been active in the OSEK/VDX movement since 1995. The company was among the first to present an OSEK/VDX-OS conforming operating system in early 1997. Today, ProOSEK is the base of BMW's standard core used in the majority of ECUs in the new 7-series. 3SOFT also participated in the definition of both the OSEK/VDX-OS and OSEKtime standards.

The I/O-Library is available as a stand-alone product and as part of the new ProOSEK 4.0. Drivers for a number of peripherals are already available. As mentioned before these need not be bought, but can easily be developed if needed. A tool for configuring the drivers is also part of the I/O-Library. This tool can be used not only for commercial drivers but also for self-written ones. When bought together with ProOSEK, the HAL configurator is integrated with the ProOSEK configurator. ■

3SOFT – 15 years of software competence

■ At the beginning of 1988 three software engineers got together to found a company focused on embedded software. It was intended that in the long term the size of the enterprise should settle down at about 20-30 employees. But matters turned out otherwise: 3SOFT went on to write an impressive story of success. The company's record up to now has been characterised by continuous and above all sound growth in turnover, workforce and customers. And the trend continues, even in these economically difficult times.

Today 3SOFT commands excellent embedded know-how that is second to none. About 150 permanent employees at the company locations in Erlangen, Braunschweig, and Stuttgart advise and develop software for a circle of customers including the leaders of the current market sectors. Agfa, Audi, BMW, Bosch (Blaupunkt), Conti Temic, DaimlerChrysler, Fresenius Medical Care, Heidelberger Druckmaschinen, Siemens A&D/Med /VDO, Techem, Visteon, VW, ZF Lenksysteme represent only an excerpt from the list of exclusive clients.

Company history

Embedded software for medical devices was the first business area addressed by the company. Since then 3SOFT has developed control and service software for Siemens Med meeting the high quality standards of the TÜV and FDA (Food and Drug Administration). Components for large medical equipment as used in angiography, computer tomography, magnetic resonance and ultra-sonics are focused.

In the early 90s 3SOFT broadened its activities into industrial automation. Siemens Automation and Drives (A & D) and Heidelberger Druckmaschinen were among the first customers. Today 3SOFT has a partnership with Siemens A & D for training and support in all matters concerning Windows CE on Mobic®, and supports Siemens' customers in the Open Platforms Program in the production and adaptation of software for operator panels.

In the middle of the 90s the European automotive industry has specified the OSEK/VDX operating system standard for electronic control units in automobiles__. Based on the collective experience with different real-time operating systems, the strategic choice fell on 3SOFT to implement this standard. The result is the ProOSEK real-time operating system, used for example in most of the control units in BMW's new 7-series.

Meanwhile In-Car Computing has become a firm pillar of 3SOFT's activities. Here, alongside OSEK, telematics plays a major role. Two engineering divisions help customers to create infotainment applications based on Windows CE and Java/OSGi.

Last but not least: in 1998 originated the youngest business area for 3SOFT GmbH, concerning itself with the technology of household automation. Thus the entire software architecture for Techems "Assisto" central domestic control system was created in the development laboratories of 3SOFT GmbH.

Specialist for standard software platforms

Transfering technologiecal know-how, professional engineering services and innovative products make 3SOFT GmbH a reliable partner. Customers of 3SOFT obtain everything from a single source, from the management of complex projects to training for the introduction of new concepts, through to the implementation and maintenance of applications. The assurance of software quality receives throughout the highest priority.

Today, 3SOFT is the expert for standard software platforms. The cen-tral concern is to assist customers to achieve cost advantages and rapid time-to-market. A current development in this direction is the company's entry as development member into the FlexRay Consortium. ■