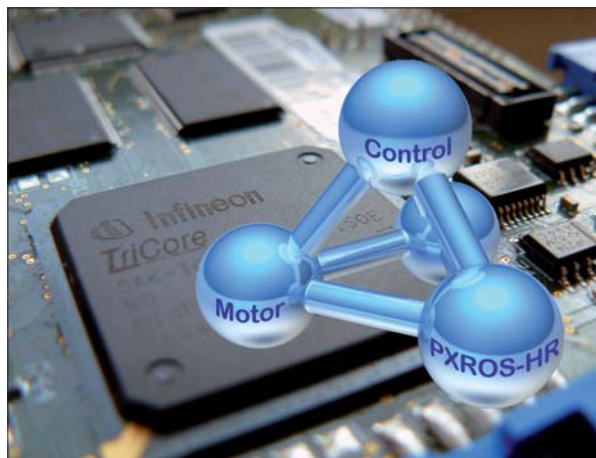


# Fine-grained memory protection guarantees safe function integration

By **Mario Cupelli**, HighTec EDV-Systeme, and **Heiko Riessland**, pls Programmierbare Logik & Systeme

*How do you combine different software modules in a single control unit without jeopardizing the safety of the overall system? The real-time operating system PXROS-HR uses the memory protection unit of the TriCore architecture to realize hardware-based memory protection, with each software module running in effect on its own controller.*



■ To meet the demand for more safety and comfort in automotive and industrial applications, a variety of different control and memory units are required. But how do you combine different software modules without risk in a single control unit? As a rule, pure software-based approaches to a solution do not offer the necessary safety, and are also expensive. To rule out interaction and possible error propagation, strict encapsulation of all the software modules to be deployed is absolutely necessary. The real-time operating system PXROS-HR, which was developed by HighTec EDV Systeme, uses the memory protection unit (MPU) of the TriCore architecture in order to realize hardware-based memory protection. It is as if each software module and each capsule were running on a controller of its own. The operating system thereby takes over the management of the MPU, and during runtime monitors the memory integrity of the capsules.

In the case of the drive application depicted in figure 1, the basic system is made up of the RTOS PXROS-HR, driver and protocols, debug monitor and a motor control unit. The TriCore architecture differentiates for software modules the privilege levels Supervisor, User-0 and User-1. In the User-1 level, accesses to some special function register (SFR) and peripheral units are still allowed; whereas, in the User-0

level neither SFR accesses nor peripheral accesses are permissible. The software modules external to the basic system, shown in figure 1 as red-framed VCUs, should typically run in the User-0 level. These so-called PXROS-HR tasks can execute services of the basic system via an application programming interface (API). As with the help of the MPU the resources have been defined by memory limits, each of these components is encapsulated. The communication between software modules also takes place solely by exchange of encapsulated messages.

The hardware-based memory protection is realized by means of the four data protection registers and two code protection registers of the TriCore memory protection unit, and is individually adjustable for the different privilege levels. By means of an upper and lower bound as well as its access right (reading, writing, executing), each protection register configures a valid “view window” in the memory. Switchover of the protection registers for the respective software module is undertaken by the PXROS-HR. Thereby any unauthorized access to a module external to the view window during the operating system runtime is detected and handled.

The typical configuration of software modules (PXROS-HR tasks) is shown in figure 2. Two of the four data protection registers configure the

sectors for constants, stack and data of a task. The two remaining registers can be used for exchanging messages. The sending of a message from Task1 and the receipt of the message in Task2 correspond to the handover of an encapsulated memory sector (object) including access right.

The protected message object can thus have any size. The memory use, in contrast to approaches that realize the protection by mapping (MMU), is minimal. As is well known, the memory efficiency achieved, particularly in embedded applications, is a determining criterion, because typically here only a small amount of RAM is available. Therefore, the PXROS-HR in combination with the MPU of the TriCore architecture guarantees a fine-grained protection which is safe, and with only approximately 5% performance loss, at the same time also highly efficient. In practice, this approach offers significant advantages for all persons involved: the system designer, the software developer and ultimately also the product manager.

For the system designer, the encapsulation of the software modules particularly means that the complexity of the software system is considerably reduced. Each software module is free of interaction and is configured with its allocated resources. At the same time, the absence

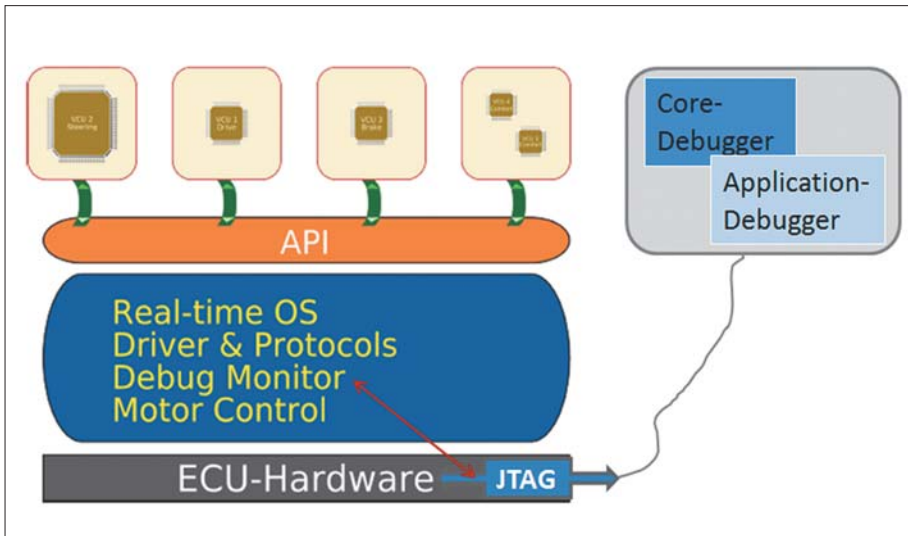


Figure 1. In the drive application depicted, the basic system is made up of the RTOS PXROS-HR, driver and protocols, debug monitor and a motor control unit.

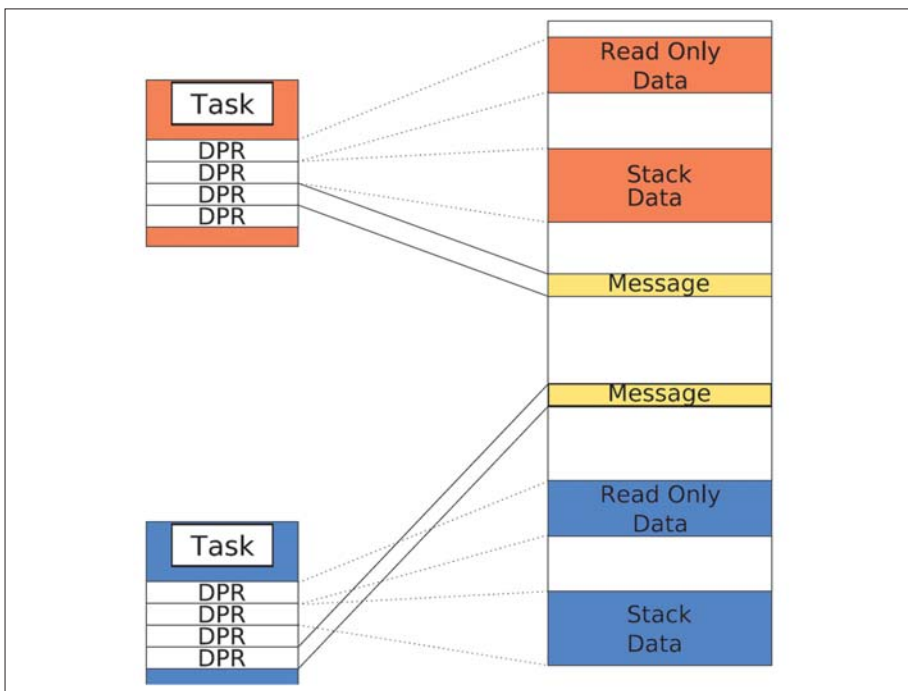


Figure 2. With typical PXROS-HR tasks, the four data protection registers configure the sectors for constants and stack of a task. The two remaining registers can be used for exchanging messages.

of interaction creates a risk-free combination of safety-critical and non-safety-critical software modules. In total, this leads to a simplified and cost-effective certification process of such applications. For software developers the encapsulation moreover brings, among other things, the advantage that, with the independence of software modules, a simple module test is possible. The adherence to the memory resources, predetermined by the system designer, for the software developer is forced by the MPU. A protection violation, caused for example by an inadvertent pointer access or transient error, does not lead to any interference whatsoever to the remaining system. By means of the encapsulated structure, safe and simple function in-

tegration is possible, because faults in software modules can be uncovered at an early stage of the development. A prerequisite is obviously that the debug tools used, as in the case of the universal debug engine (UDE) from pls Programmierbare Logik & Systeme, also observe the mechanisms used and support two essential operating modes.

The first operating mode uses a JTAG debugger for debugging of the PXROS-HR kernels. The memory protection unit employed by the operating system is generally used for the realization of a maximum of four hardware code breakpoints and further data breakpoints. The MPU resources must be shared between de-

bugger and PXROS-HR. The current implementation enables the setting of two code breakpoints in the debugger. The remaining memory protection unit is used for memory protection, which is realized by the operating system. The second operating mode enables the debugging of currently up to two reloadable PXROS-HR tasks. Because a complete stop of the hardware here is not wanted, the application debugging takes place via a debug monitor, which is integrated in the target operating system. The monitor encapsulates task-specific breakpoints, context handling and call stacks for the debugger which is to a large extent transparent and addressable via a gdb compatible command interface. Ethernet and serial interfaces serve as standard connection to the target monitor. However, JTAG as communication channel is also usable with the universal debug engine and an enhanced monitor. The latter is an advantage, because firstly Ethernet is not a typical interface for TriCore targets and secondly both the kernel debugging and the application debugging can take place, if required, via a JTAG connection.

Ultimately, product managers also benefit from the fact that a fault in a software module cannot jeopardize the integrity of the entire system. If the cause of the fault can be definitely associated with a specific software module, the question of product liability can also be clearly and conclusively regulated.

Summing up: the encapsulation of software modules by means of hardware-based memory protection does not only lead to appreciable reduction of the test and certification efforts, but also enables simple and safe function integration, and protects, in the event of system crash or damage, against possible unjustified product liability claims. However, the development of such complex systems presumes, as described in the example of the operating system PXROS-HR from HighTec and the universal debug engine from pls, that all tools used support the encapsulation of software modules. ■