High Reliability and Performance of OS-Compatible Systems for Embedding Purposes

ABE Tsuyoshi, SAKAI Junji, TORII Sunao

Abstract

It is predicted that in the future there will be a need for embedded equipment that will feature very advanced information processing and high-functionality to enable very complicated processing. This is as important as the traditional requirements for high reliability and high responsiveness. This paper proposes a dual OS that fulfills all of the above requirements by combining a real-time OS for embedding and an SMP OS for use as the server. Its effectiveness is demonstrated by packaging it with the MPCore SoC multi-core processor for embedded applications.

Keywords

multi-processor, embedded system, real-time OS, high-function OS, responsiveness, reliability, functionality

1. Introduction

Since recently developed embedded systems such as the cellular phone and car-mount devices should be capable of complicated information processing, they require high functionality in addition to the traditional requirements for embedded systems of high responsiveness and high reliability. However, a real-time OS such as ITRON does not have high functionality because it has to secure responsiveness and reliability by providing the minimum number of functions mainly targeted as equipment control type processing. On the other hand, a high-function OS targeted at advanced information processing, such as Solaris, Linux or Windows, cannot secure enough responsiveness and reliability compared to a real-time OS because improvement of the OS functionality complicates the packaging, degrades the predictability and makes it impossible to achieve the high responsiveness and reliability required for embedded systems. As a result, there is for the present no OS that is suitable for embedded systems with a high functionality requirement (**Fig. 1**).

In consequence, to implement an embedded system with practical high functionality, it is necessary to develop a technology that allows multiple OSs with different properties to coexist in a single system.

2. OS Coexistence Techniques

One of the issues in running more than one OS on a single system is how to eliminate competition between resources,

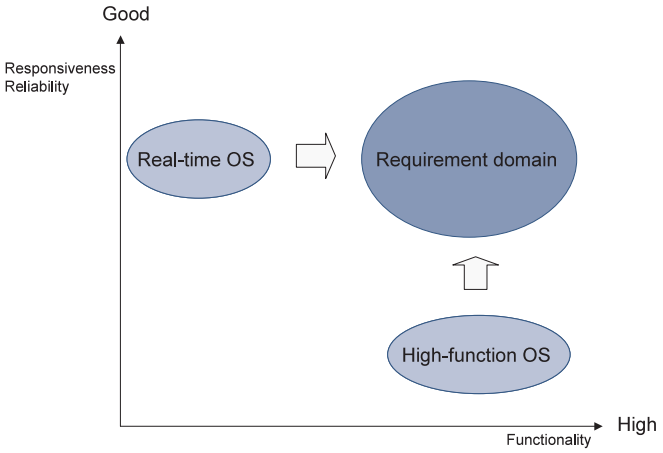

Fig. 1 Requirements for embedded systems.

which include the CPU, the memory layer including the cache, the bus and the devices connected to peripherals. The techniques for implementing OS coexistence can be classified roughly into the following three categories according to the methods of their arbitration; 1) hybridization technique for running all OSs except for one on an OS; 2) virtualization technique for multiplexing hardware for the OSs; 3) native technique by introducing a multi-core processor and applying correction of OSs for arbitration (**Fig. 2**).

Table shows the features of these techniques. The performance is highest with the native technique with which resources can be accessed directly from the OSs, the packaging cost is lowest with the hybridization technique that uses an existing OS for competition management, and the universality is highest with the virtualization technique that conceals the hardware

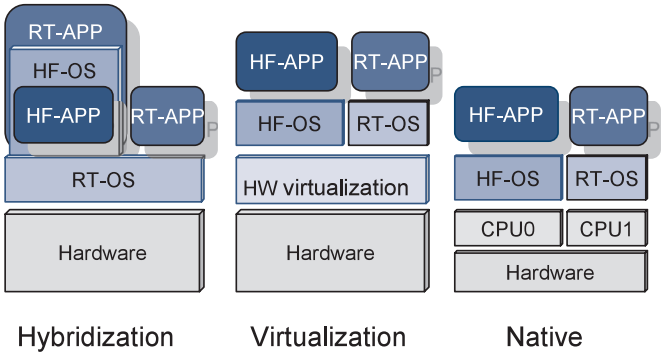

Fig. 2 OS coexistence techniques.

Table Comparison of OS coexistence techniques.

Technique	Advantages	Disadvantages
Hybridization	Low costs thanks to batch management of competition resources using existing OS.	The entire performance and system are determined depending on the managing OS. The performance of the embedded OS may degrade.
Virtualization	High universality thanks to avoidance of competition in low-level layers.	The performance and reliability tend to decrease due to the software processing for hardware multiplexing.
Native	Little performance degradation thanks to direct resource access.	A multi-processor is necessary. High dependency on the hardware reduces the universality.

layer. For embedded systems with which malfunction is sometimes fatal, OS separation using software becomes a factor degrading the reliability. At NEC, we have determined that the native technique is most suitable for embedded systems considering the suitability of the above coexistence techniques to embedded systems. Moreover, it has already become realistic to incorporate multiple processors in an LSI for embedded systems thanks to the progress of semiconductor integration technology.

3. OS-Coexisting SoC Environment for Embedding Applications

In order to build a practical embedded system using the native technique the technology for OS coexistence is not by itself enough, a technology for maintaining high responsiveness and reliability proper to embedded systems is also necessary. From this viewpoint, we determined the following issues and attempted to solve them from the aspects of both software and hardware.

The OS coexistence issues include; 1) arbitration between devices that share a single physical resource; 2) the provision

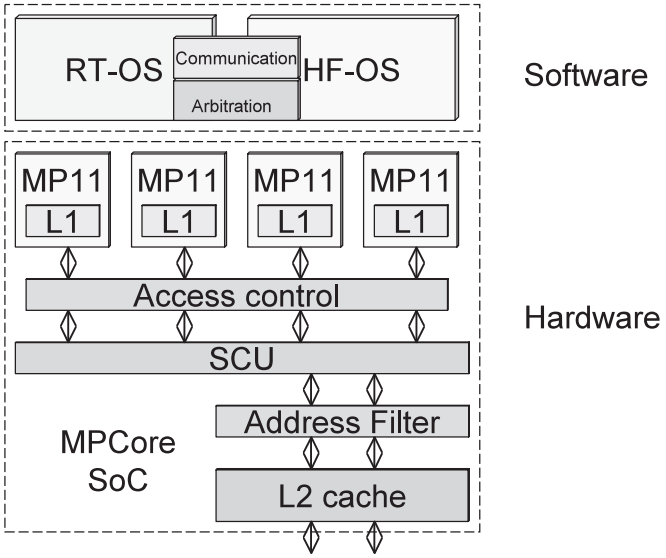

Fig. 3 Configuration of OS-coexistent system.

of a fast on-chip inter-OS communication mechanism. We adopted software-based solutions for these issues (**Fig. 3** upper part).

In this solution, we divided the OSs into three mechanisms according to purpose. In the arbitration layer, we packaged mechanisms for the initialization of shared devices, setup of access-restricted hardware, startup of OSs and arbitration between OSs in order to solve the device arbitration issue raised in 1). In the communication layer, we packaged the mechanism for inter-OS communication in order to solve issue 2). Here, we used the same inter-processor communication mechanism used in the multi-core system to achieve communications at a high speed. We installed an OS that is modified to solve the issue of competition between shared devices using the arbitration layer in each CPU to solve the overall issues of OS coexistence.

On the other hand, the issues in meeting the requirements for embedded systems include; 3) prevention of performance degradation due to inter-OS interference; 4) reliability in the case of OS runaway. We adopted hardware-based solutions for these issues (Fig. 3 lower part). For issue 3, we attempted a solution by utilizing the interrupt distribution mechanism and flexible cache setting facility of the MPCore multi-processor for embedded use, and succeeded in reducing the effects on performance from other OSs thanks to an independent interrupt setting for each OS and L1 cache coherency setting. For issue 4), we solved this problem by proposing an access control function that can restrict access to specific registers in the

MPCore of each CPU and an address filter that restricts the accessible address space per CPU.

Such measures made it possible for us to build a system that can always maintain high responsiveness and reliability of the real-time OS even in the case of runaway of a high-function OS running on an adjacent CPU.

Specifically, it was NEC Corporation that proposed such a system architecture for embedded applications and it was NEC Electronics Corporation that designed and packaged the MPCore SoC for embedded applications in collaboration with ARM Ltd.

4. Verification

We built a dual-OS environment with TOPPERS/JSP and SMP Linux, mounted a demonstration system for car-mount use in this environment, and demonstrated that the OS coexistent environment for embedded systems discussed herein can meet all of the responsiveness, reliability and functionality requirements simultaneously.

Fig. 4 shows the configuration of the demo system, which runs a road condition recognition application by linking the image processing on TOPPERS/JSP and the recognition processing on Linux. The application also displays a map showing the current vehicle position and related information on a browser running on Linux. In addition, a performance monitor functions as the performance display and irregularity monitoring system on each OS. It was thus proven that the OS coexistence is practical and possible when using this configuration.

With regard to the reliability of the real-time OS, we attempted to use the mutual OS monitoring function mounted in the performance monitor for detecting abnormal load and deadlock in Linux, and forced the termination of the CPU used by the Linux as a trial. As a result, we were able to confirm that the TOPPERS/JSP does not lose stability even when Linux is malfunctioning and that a forced termination of Linux does not affect the operations of TOPPERS/JSP.

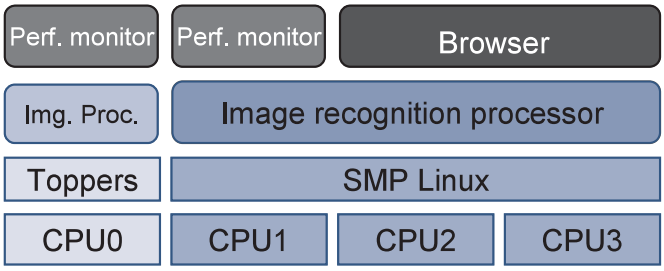

Fig. 4 Configuration of demonstration system.

Fig. 5 View of demonstration.

The response time (time from generation of an interrupt to startup of the interrupt handler) was 3μs on average and 6μs in the worst case. As it was maintained as high as 6μs on average/15μs in the worst case even when maximum load (100% bandwidth) was applied to the bus under concurrency of the two OSs, we were able to confirm that the system was capable of maintaining a high responsiveness. In addition, due to the fact that the worst value trend did not change at or above a certain load, we could also confirm that a certain performance can be maintained thanks to the mechanism (round robin mechanism) ensuring fair arbitration of requests from the CPUs to a bus (**Fig. 5**).

5. Conclusion

In the above, we propose an overview of the requirements for next-generation embedded systems and describe the efforts made at the NEC Group for the multi-core-based SoC system technology for use with them.

At NEC, we are able to make full use of the hardware and software technology of the NEC Group that has been developed up to the present time, so that we can continue to make proposals for embedded systems with higher reliability, higher performance and higher functionality for our customers in the future.

Authors' Profiles

ABE Tsuyoshi
SoC Design TG,
System IP Core Research Laboratories,
NEC Corporation

SAKAI Junji
Principal Researcher, SoC Design TG,
System IP Core Research Laboratories,
NEC Corporation
Member of IPS (Information Processing Society of Japan) and IEEE-CS

TORII Sunao
Principal Researcher, SoC Design TG,
System IP Core Research Laboratories,
NEC Corporation
Member of IPS