RTOS Evolution and Hardware Microkernel Implementation Benefits

Ahmed Karim Ben Salem and Slim Ben Saoud
LECAP-EPT / INSAT, Centre Urbain Nord, Tunis Cedex, Tunisia

Due to the importance and complexity of real-time applications, the demands on real-time platforms increase every year, which motivates moving these applications onto multiprocessor hardware system on chip (MPSoC). This paper deals with different methods used for efficient handling of real time design projects leading to their successful completion, not just completion on time, but completion with as little silicon surface as possible. In the following pages, we outline the evolution of Real Time Operating System (RTOS) architecture and their limitations, compare different approaches and investigate the gain using the hardware implementation with microkernel structure for these RTOS that will improve both performance and design time.

1. Introduction

Great progress made by the semiconductor industry, smaller process geometries and higher densities have all provided design engineers in the field of real time systems means to create complex and high-performance System on Chip (SoC) designs.

Furthermore, trends in chip design - the new SoC platforms and the introduction of software-to-hardware tools for FPGAs- have improved the practicality of these devices as software-programmable computing platforms. They have offered the opportunity to bring the embedded Real Time Operating System (RTOS) onto the FPGA to improve both performance and design time.

To fully exploit such chips, different approaches are used to obtain better performance in RTOS such as the use of special purpose hardware, for example, the Real-Time Unit (RTU) [5], which contains RTOS services implemented in hardware and the use of the microkernel architecture to speed up system-calls, task management, inter-process communication for a real-time kernel [3][7].

2. RTOS Scalability

The main task of an RTOS is to manage the resources of the computer such that a particular operation executes in precisely the same amount of time every time it occurs. An RTOS provides a complete software support, while a Real Time Kernel (RTK) is intended to be used with or without association with an OS.

In an RTOS, the tasks are generally complete programs. While in a RTK, the tasks are rather functions of the same program. In general, the RTOS are used in the computers to manage significant data amount, but the RTK are used in embedded calculators.

Today's RTOS has the ability to scale in order not to use more resources than necessary. The most obvious resource is the ROM and RAM footprint. The purpose of the software scalability is most often to reduce an extensive configuration by reducing the amount of functionality supported. This concerns, for example, the size, the number of tasks, synchronisation, and communication mechanism.

Two RTOS generations appeared since the emergence of the commercial application. The use of the *first RTOS generation*, a category in which we usually classify as *VxWorks*, *pSOS* or *VRTX*, is trivialized today. But this generation suffers from several limitations. First of all, they are unable to benefit, in an ideal way, from the memory partitioning mechanisms offered by the microprocessors. Indeed, with this type of OS, all tasks share a single addressing space. In addition, they are not adapted to the multiprocessors requirements for the distributed real time systems.

About fifteen years ago, the arrival of *a second generation* of the commercial RTOS in the market reduced some of these limitations. Oss, like *OSE*, *QNX* or *LynxOS*, support the MMU (Memory Management Unit) specific to the principal 32 bits processors. They can, therefore, manage applications claiming a certain level of faults tolerance since different tasks with the MMU can function in separate and memory-protected space. In addition, the techniques of direct message passing between tasks, supported in standard by *OSE* or *QNX*, which are independent of the processors location and the network type connecting them, simplify considerably the development of distributed applications or high availability software.

The RTOS hardware implementation has reduced considerably some limitations of software RTOS and presents a better performance.

When the hardware kernel was introduced into multiprocessor systems, it was called real-time unit (RTU). It moves the scheduling, inter-process communication (IPC) such as semaphores and time management control, for instance, the time ticks and delays from the software OS-kernel to hardware. More functionality and improvements were added.

The previous work of [5], [7] and [8] has shown the benefits of having the RTU activities implemented in hardware. The system overhead is decreased resulting in improved predictability and response time, the CPU load and memory footprint is reduced, and less cache misses are seen.

In [1] and [4], the RTU was used in a research project in multiprocessor systems called Scalable Architecture for Real- Time Applications (SARA). The SARA-system is based on the idea to incorporate as many parts of a RTOS into hardware as possible. The scalability of the SARA-system could be used in the transition from a single processor system into a multiprocessor system. The RTU handled the scheduling of the system.

Microkernel Approach

RTOS used in today's embedded systems need to reflect the modularity of the underlying hardware and the increasing demand for adding and replacing functionality in the system. A RTOS structured with the microkernel concept fulfils these requirements [3].

Monolithic kernels (Fig. 1.a) are large in code size with complex structure which makes them difficult to change and maintain without affecting other parts of the kernel.

The microkernel approach (Fig. 1.b) is based on the idea of only placing essential core RTOS functions in the kernel, and others functionalities are designed in modules that communicate through the kernel via minimal well-defined interfaces. So, only a minimal part of the OS runs in kernel mode, whereas all applications run in user mode. A microkernel's main function is resource management.

The OS kernel contains only a small core of fundamental services, such as timers, messages, and scheduling. All higher level services and programs – drivers, file systems, protocol stacks, and user applications – run outside the kernel as separate, memory-protected components.

A microkernel OS has a loosely layered structure with client-server message passing communication between the layers, a well defined communication mechanism that allows programs to exchange data while remaining safely isolated from each other.

An example of a true microkernel is L4 [6]. Several others microkernels have been developed such as ChorusOS, EROS, Nucleus, and VxWorks.

The first generation of microkernels suffered from poor performance, which led to bad reputation of this kernel structure. In the second-generation microkernels, performance has increased and it is no longer a problem [2]. The client-server message passing idea of the microkernel structure results in more context switching compared to monolithic systems [2]. Implementing the real-time kernel in hardware, following the microkernel structure approach, results in a microkernel with the same or better performance as monolithic structured systems, but without the latencies that first generation microkernel systems has been suffering from.

The work of [7] shows that a microkernel structured kernel in hardware can be ported to an existing monolithic RTOS, and how it affects the performance of the system-calls.

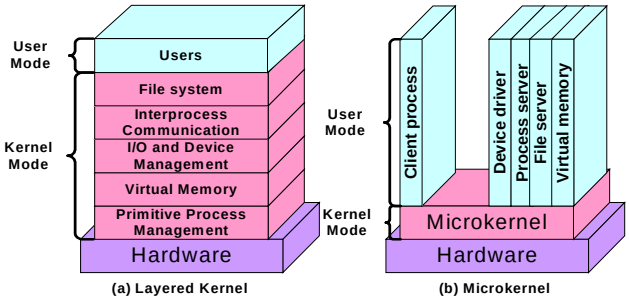

Fig.1: Layered vs. Microkernel architecture.

3. Conclusions and future work

The RTOS overview presented in this paper showed that hardware RTK introduced into MPSoC has several benefits as compared to the pure software RTOS system. Moreover, microkernel RTOSes offer inherently greater modularity than conventional OSes, add flexibility to the system, enabling the stringent testing, extreme fault tolerance and dynamic upgradeability that many battlefield systems require.

We are currently working on a CoDesign of hardware RTOS, using the new SoC trends, dedicated to digital control systems.

References

[1] L. Enblom and L. Lindh, "Adding flexibility and real-time performance by adapting a single processor industrial application to a multiprocessor platform". in 'Parallel and Distributed Processing', Proceedings of the 2001 EUROMICRO Workshop, Mantova, Italy, p. 487, February (2001).

[2] H. Härtig, M. Hohmuth, J. Liedkte, S. Schönberg and J. Wolter, "The performance of µ-Kernel-based systems", in Proceedings of the 16th ACM symposium on Operating Systems Principles, Saint Malo, France, p.66 (1997).

[3] J. N. Herder, "Towards a true microkernel operating system", Thesis in Computer Science, VRIJE University Amsterdam, February (2005).

[4] T. Klevin and L. Lindh, "Scalable architecture for Real-Time applications and use of bus-monitoring". in Proceedeings of RTCSA'99, p. 208-211, December (1999).

[5] J. Lee, J. V. Mooney III, K. Ingström, A. Daleby, T. Klevin and L. Lindh, "A comparison of the RTU hardware RTOS with a hardware/software RTOS", in Proceedings of the ASP_DAC 2003, Design Automation Conference, p. 683-688, January (2003).

[6] J. Liedtke, U. Dannowski, K. Elphinstone, G. Lieflander, E. Skoglund, V. Uhlig, C. Ceelen, A. Haeberlen and M. Volp. "The L4Ka vision", System Architecture Group, University of Karlsruhe, Germany, April (2001).

[7] S. Nordström, L. Lindh, L. Johansson and T. Skoglund, "Application specific real-time microkernel in hardware" (2005).

[8] T. Samuelsson, M. Åkerholm, P. Nygren, J. Stärner and L. Lindh. "A comparison of multiprocessor real-time operating systems implemented in hardware and software". International Workshop on Advanced Real-Time Operating System Services (ARTOSS), Porto, Portugal (2003).