Scalable Architecture for Real-Time Applications And Use of bus-monitoring

Tommy Klevin and Lennart Lindh
Department of Computer Engineering (IDT), Mälardalens Real-Time Center (MRTC)
Mälardalens University, Sweden
E-mail: tkn@mdh.se, llh@mdh.se

Abstract

The lifecycle for industrial applications are becoming shorter, the application complexity increases, performance is to low, fault tolerance is required, reuse of components is desired and the developer require strong verification tools to cut down the verification phase. As the problem increases with respect of longer development time and higher quality requirements from the customer, it becomes increasingly important to examine flexible and scalable parallel processing for complex real-time systems. This is the motivation for running the research project ***SARA*** *(**S**calable **A**rchitecture for **R**eal-Time **A**pplications). The first SARA system is now running with a vision system connected to an industrial robot (ABB Robot).*
The system-busses are important resources in a computer-system. Today there are no methods to ***monitor busload during runtime****, in this paper we discuss a simple method of how to do this.*

Keywords: *real-time kernel, real-time system, architecture, bus monitor and multiprocessor systems*

1. Introduction

In the last decade, the complexity of real-time systems has increased. The systems must nearly always have their hardware/software architecture redesigned to increase their performance, get better fault tolerance, etc. New powerful processors alone are not sufficient to achieve high performance and flexibility of the control systems (real-time systems).
Today it is usually the inadequate performance of the real-time system, the inflexibility of changing the hardware/software architecture, the complexity of the solutions and the weak debugging tools for verification that causes *time to market problems.*

One bottleneck in computer systems today is the performance of the busses in the system. If the operating system could get information about actual busload it would help the operating system to schedule tasks more efficiently.

Multi-processor systems like **SARA** tend to be very complex and therefore it is hard to understand how transactions during runtime will affect the system. As the real-time kernel, named Real Time Unit (RTU) [1] in SARA, keeps information about all tasks we have a suggestion to add a new parameter that would give the kernel information about the **busload** in the system. We belive this parameter could help the scheduler to utilize the system resources better.

2. Problem motivation and state-of-the-art

Most Hardware/Software architectural implementations of complex control system designs are carried out by a number of static coupled processor units, integrated on a PCB (Printed Circuit Board) or on a separate PCB with a standard bus and dual port RAM.
SARA (**S**calable **A**rchitecture for **R**eal-Time **A**pplications) is a dynamic coupled processor system (flexible processor system) and is not dependent on the PCB (Printed Circuit Board) or on the software architecture. There is always a limit of what a dynamic system can handle. If a system is building on a parallel bus, the bus will be the bottleneck when to many processors are connected to it.

The definition of a static coupled processor system is; changes of the system requirement needs a redesign of the application software, of the PCB and a major change of software communication and synchronization between the processors. A change of the requirement for a dynamic coupled processor system can be redesigned without any changes on the PCB or in the software for tasks, communication and/or synchronization.

It is not motivated to develop static coupled processors systems, since the performance requirement from the application is growing faster than the processor developer can accomplish.

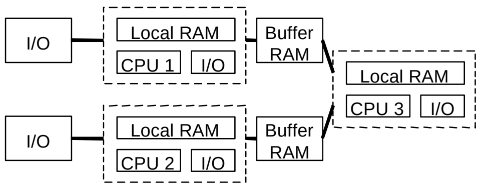

Figure 1: Static coupled multi-processor system

The problem with static coupled systems:

- Statically designed PCB (Printed Circuit Board) is not a flexible solution.
- Processors are often communicating via dual port memory (buffer). The communication and synchronization are not trivial problems and often put a heavy load on the processor.
- The software architecture is statically divided into different processors, often with different software developing paradigms, for example a signal processor and a RISC processor.

Research into real-time multi-processor systems is in progress in such projects as: MARS [2] (Maintainable Real-Time System) in Austria, RTU [1] in Sweden and SPRING [3] in USA. SPRING indicates that a dynamic real-time system can be built as a multi-processor system, while MARS presents a statically distributed system for safety critical applications. RTU has proved that complex functions can be implemented in multi-parallel hardware. It can be used as a complex controller of a multi-processor system with a speed increase in the order of 300 % (or more) for the real-time service in the kernel [4].

A trend towards MIMD (multiple instruction multiple data) multi-processor architectures has been observed during recent years. Multi-processing benefits include increased flexibility and lower cost/performance factor.

3. System Architecture of SARA

The system architecture supports a simple design paradigm and a simple verification environment.

SARA is divided into application, base system and hardware platform.
Applications is designed with an object-based approach. Objects is divided into three base classes; shared, server and base object. The base system is a collection of a communication/synchronization system for the application (IPC), verification/analyze system and resource/time handling (RTU). The base system is implemented both in hardware and software classes.
The hardware architecture is divided into local CPU board, bus arbitrator, global RAM, I/O and an RTU. The RTU is a hardware object in the base system.

RTU - a class in the base system
Many real time control systems use an application, which is controlled by a real-time operating system for executing processes. To improve the performance of a real time control system, the processor clock frequency can be increased. Sometimes this is not sufficient and a co-processor can be used instead. The co-processor (we call it an RTU) is not a standard processor, but a special purpose hardware performing real time operating system functions. Different real time operating system functions have successfully been implemented into hardware the last 10 years. The scheduling algorithms of the RTU are preemptive, non-preemptive or mixed. When the RTU uses preemptive scheduling, it uses an interrupt to signal the application processor to start a context switch. The scheduler algorithm of the RTU can also *load balance processors* (more information about RTU see [1]).

The diagram below is the result of a benchmarktest where a commercial RTOS was compared to the RTU. The diagram shows the administrative load, i.e. the time spent in the kernel. As the diagram shows the RTU adds no such load while the commercial RTOS has an administrative load of 32% with 1ms clocktick and <5% with 32ms clocktick.

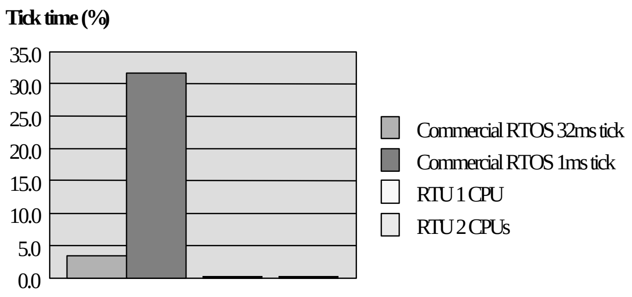

Figure 2: Kernel workload of commercial RTOS and RTU

IPC - a class in the base system
The application software (task or server class) connects to an IPC bus, it can be seen as a virtual bus. The IPC bus

contains 32 slots and each slot can handle 32 messages in a queue. A slot can be owned by a task (we call it a server object). The slot of the processors can be allocated in two ways:

1) One slot is allocated to one processor
2) One slot is allocated to two or more processors, which means it is scheduled between the processors.

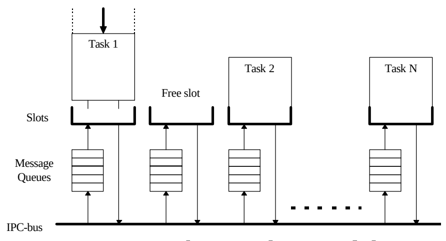

Figure 3: The IPC bus model.

In the IPC bus model (see fig.3), slots are RTOS resources that can be allocated by tasks. Each slot consists of a message queue, which holds the priority of messages and references to the messages, stored in a message buffer. Every message has a priority, which is set by the sender. The messages in the queues are sorted by their priority or in FIFO order. A task can inherit its priority from the messages (to avoid priority inversion). A sender task can use time-out constraints on full queues, and a receiver task can do the same on empty queues, e.g. a receiver task can be set to wait a specified time for a message.

There are four **message-types**. These are:

• Asynchronous messages.
• Synchronous messages.
• Broadcast messages.
• Multicast messages.

The system is implemented in both software and hardware, see figure 4. Some parts are implemented in hardware to achieve higher performance and/or to attain a simpler solution.

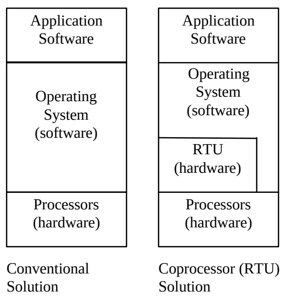

Figure 4: Overview of software and hardware implementations in SARA

4. Hardware Architecture of SARA

The hardware platform in the SARA- system is a Compact PCI system (CPCI) [5] with eight slots. In these slots CPU-boards can be inserted. A CPCI-system has two kinds of boards. In the first slot there is always a 'system-board' while the other slots holds 'non-system boards'. The system-board handles arbitration, clock-distribution, etc on the backplane. In SARA it is possible to use 1-8 CPU-boards. Minimum requirement is one board, the system-board. If more CPU-power is needed up to seven non-system boards can be inserted. In SARA all CPUs have a local memory and there is no special global memory. If global memory is needed, a global area can be defined on any CPU board. In the SARA-system, global memory resides on the system board. As the architecture allows tasks to migrate between CPUs global memory is used for Task Control Blocks (TCB) etc for these tasks. Tasks can be allocated to a certain CPU as well and in this case the TCB is stored in local memory.

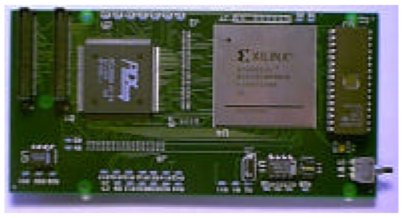

Figure 5:Picture of an RTU-PMC board

All boards in the system have a local PCI bus and all boards are connected to the CPCI backplane.

The Real Time Unit is attached to the local PCI bus on the board in the system slot from where it can communicate with all CPUs in the system.

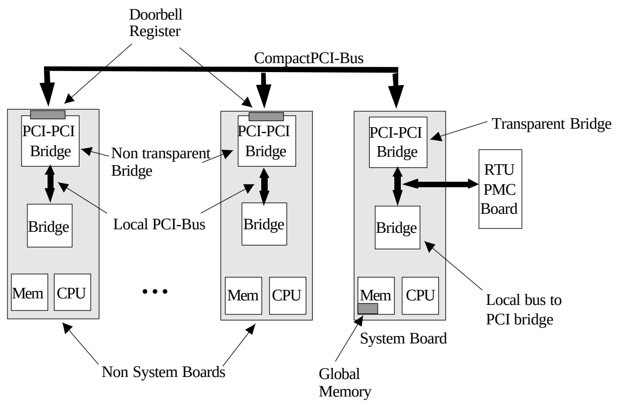

Figure 6: Block diagram of SARA-System

5. Use of bus monitoring

The present implementation is a PCI-bus monitor implemented in an FPGA (Field Programable Gate Array). To measure the load on the PCI-bus only a few signals has to be involved. The monitor listens to the signals FRAME, IRDY, DEVSEL and TRDY on the PCI-bus [6]. A combination of these signals makes it possible to decide the utilization of the bus. In the present implementation these signals are sampled during 50000 clock-cycles at 33MHz. When a sample is complete an interrupt is generated to a PC and the result is transferred via the parallel port. The load is presented on the display in percentage (0-100%) .

Busmonitors available on the market are used to debug hardware and software while the bus-monitor in this article is supposed be integrated in a system to measure bus-load in runtime.

The goal with monitoring the busses is to get a better control and observability of the system. To obtain this, the information from the monitors has to be integrated with the scheduler in the system. We believe that this information can be used to make the resource utilization more efficient.

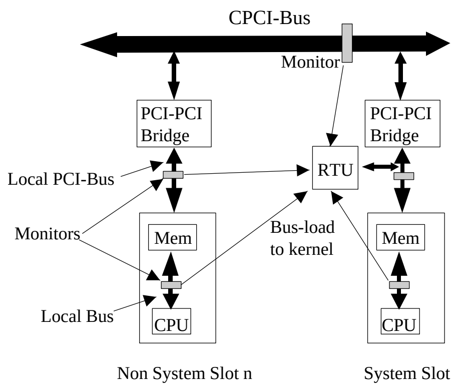

Figure 7: Monitors sending load info to Real-time scheduler

Conclusions

A static multiprocessor system is often complex and it is costly to redesign the system, compared with a flexible multiprocessor system like SARA.

The IPC communication interface is a very attractive solution; it is very easy to write interface and to make connections between the concurrent objects. In addition, the priority inheritance of messages solves the inversion problems.

When one critical function is implemented into a hardware unit, the response time and time gap between the best and worst execution time decreases. As an example, the period time for clock tick is one microsecond for the hardware accelerator (RTU) and that is about 1000 times faster than software solutions.

Simple bus-monitors can easily be implemented in a system and the actual bus-utilization can either be sent to a PC-display or the information can be sent to the operating system and be used in the scheduling algorithm.

Future expansions and works:
We have plans to integrate the information about the bus-load into the RTU.
Today most of the IPC-communication is implemented in software. To get higher performance and more predictable IPC all primitives will be implemented in hardware. The only software needed will be the interface to the IPC-calls.

Acknowledgements

These projects are sponsored by KK-Foundation, internal founding by Mälardalen’s University College, ABB Robotics and Ericsson Utvecklings AB.

References

[1] LLindh, J Starner and J Furunäs, From Single to Multiprocessor Real-Time Kernels in Hardware, IEEE Real-Time Technology and Applications Symposium, Chicago, May 15 - 17, 1995
[2] H.Kopertz, R. Zainlinger, G. Fohler, H. Kantz, P Puschner, W. Shutz. Institute fur Technische Informatic, Technische Universität Wien, Treitlstr. 3/182 A-1040 Vienna, Austria. An Engineering Approch to Hard Real-Time 1991
System Design
[3] J. Stancovic and K. Ramamritham. Hard Real-Time Systems. ISBN 0-8186-0819-6, pages 371-382
- J. Stancovic, D. Niehaus and K. Ramamritham. Spring net: An Architecture For High Performance, Predictable and Distributed Computing. Computer Science Department, University of Massachusetts.
[4] L. D. Molesky, K. Ramamritham, C. Shen, J. A. Stankovic, G. Zlokapa, ”Implementing a Predictable Real-Time Multiprocessor Kernel - The Spring Kernel”, 1990
[Lawson98] Harold W. Lawson, "Salvation from System Complexity," Computer, Vol. 31, No. 2, February, 1998, pp. 118-120.
[5] PCI Industrial Computers Manufacturers Group CompactPCI Specification Revision 2.1
[6]Periferal Component Interconnect Specification 2.1