Flexible Multiprocessor computer Systems

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

Abstract

Computer Graphics applications require a high perform computer system. The lifecycle for graphical applications are becoming shorter, the application complexity increase, performance is to low, predictability is needed for a good qualities, reuse of component is desired and the developer require strong verification tools for cut down the verification phase. As the problem increases in 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**caleable **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).
This paper defines today's problems, discusses the state of the art, presents the research project SARA, and presents some results and conclusions from the first implementation.

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).
The gap between the processor performance and the application needs is increased. The performance increase of 60% per year of the processors is decreased of the latency decrease from the memory of only 7% per year[Hennessy95]. 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 cause *time to market problems*.

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 or on a separate PCB with a standard bus.
It is motivated to develop static coupled processors systems, since the performance requirement from the application is growing faster than the processor developer can accomplish.

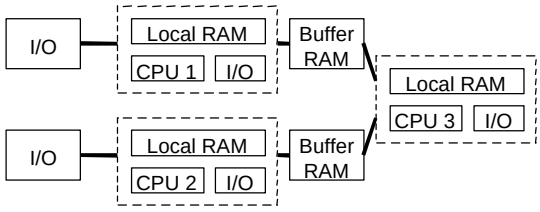

Figure 1: Static coupled multi-processor system

The problem with static coupled system:

- 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.

This makes the design process of the whole system very complex.

The first multi-processor systems date back to the sixties. Now multi-processor systems have reached less expansive systems such as workstations (for example [SUN]) and PCs). Such systems are not always suitable for real-time systems, as they are not sufficiently safe, reliable, predictable, cost effective, etc. Research into real-time multi-processor systems is in progress in such projects as: MARS [Kopertz91] (Maintainable Real-Time System) in Austria, RTU [Lindh95] in Sweden and SPRING [Stancovic] 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 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 [Molesky90].

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. The SARA Approach

The new approach is defined by the following design goals:

1. *Efficiency, Performance and flexibility,*

High performance is one of the most important goals in a multi-processor system.

To achieve a high performance the system must be based on state-of-the-art, commercial, standard microprocessors, busses, and different hardware accelerators etc. Unfortunately, this means that cache, pipeline, etc represent predictability problems. Deterministic high performance processor architecture is possible to develop, but it is difficult to compete with the big processor companies.
A architecture for *Processor Scalability* is a flexible way to increase the performance for the application. The architecture should be a scaleable open system with no theoretical limit [Dal94]. A node can consist of hundreds of processors and can be hierarchically structured. The system can be configured as a mix of loose and hard coupled system. There is a need for an "intelligent" scheduling algorithm, that can take into account time cost for overhead time, such as task switch, data transferring, priority, resource allocation, etc, and also control the prefetching of data ([Furunäs97] and [Stärner96]).

2. *Predictability,*

The software and hardware should be partly predictable. In a complex system, often 80-90 % of the tasks have soft deadlines (non-critical) and 10 % have hard deadlines (critical tasks).

3. *Observability and controllability*

The verification allocate 50-75% of the hole development time. Easy debugging and performance monitoring is also a important goal to reduce the development time. It should be possible to verify the behavioural and the time requirements in a computer model, without the hardware [98Mohammed].

4. *Low Hardware and Software Overhead (simplifications),*

The non-productive software and hardware should be minimized [Lawson98]. Simple solutions are important aspects when the design decisions are taken. The base system and the hardware platform should be as simple and small as possible.

5. *Component oriented design,*

Component design is one important goal for decreasing the development time. The system should easily handle components, i.e. software or hardware components. The design paradigm will rest on an object-based software/hardware design and a

priority inheritance based communication protocol. To reuse different interface standards, an "adapter" in hardware or software is used.

6. *Fault Tolerance*

Many real-time applications are safety critical. They must function at least partially under severe disturbance conditions [Kopertz91]. Reliability and a high degree of availability are crucial in meeting today's quality requirements. In addition, software reliability and robustness with respect to third-party software are required. Problems as overload and failures must be handled in an dynamic, adaptive way.

4. System Architecture of SARA

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

The system is divided into application, base system and hardware platform.

The application is designed with an object-based approach. The object is divided into three base classes; shared, server and base object. The base system is a collections 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.

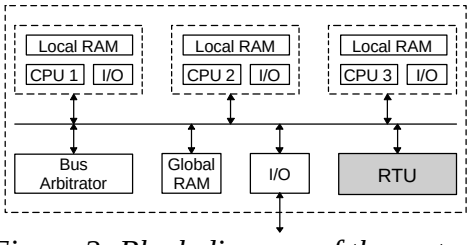

Figure 2: Block diagram of the system

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 so 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 [Lindh95]).

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 has 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.

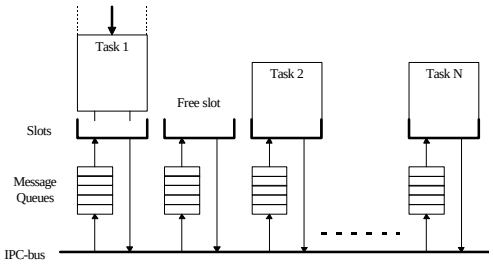

Figure 3: The IPC bus model.

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:

- Asynchrony messages.
- Synchrony messages.
- Broadcast messages.
- Multicast messages.

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

6. Hardware Architecture

The hardware platform in the SARA-system is a Compact PCI (CPCI) [PICMG97] system with eight slots that can hold CPU-boards. In a CPCI system, there is always a special 'system-slot'. This slot has a special CPU-board that handles the arbitration, clock-distribution, etc on the backplane. All other slots have 'non-system boards'. The CPCI architecture fits well into the idea of a flexible and scalable system. The basic idea of having a scalable hardware platform is that it is easy to add more CPU power when needed. A traditional system is usually made up of a design that is static. Consequently, it is not possible to do any changes to it unless it is redesigned.

In a CPCI-system more CPU-power can be added just by inserting more CPU boards. Boards can even be inserted and deleted with the power on (hot swap). In a CPCI system, all CPUs have a local memory and no special global memory. If a 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. The global memory is used for Task Control Blocks (TCB), global variables and stacks. The code for tasks is stored in the local memory. All boards in the system have a copy of the task code. This means that a task can be executed anywhere in the system. The decision where a task will be executed is taken by the RTU.

Figure 4:Picture of a RTU-PMC board

There are two kinds of PCI-busses in the system. All boards in the system have a local PCI bus and all boards are connected to the CPCI backplane. The local PCI bus is connected to the CPCI bus on the backplane through a PCI-PCI bridge. The system board has a transparent bridge, while non-system boards have a non-transparent bridge. The transparent bridge makes an address on one side of the bridge appear as the same address on the other side. The non-transparent bridge can remap an address from one side of the bridge to another address on the other side. The advantage of non-transparent bridges is that address collisions can be avoided on the backplane and that all boards can use its full address-range on the local PCI bus.

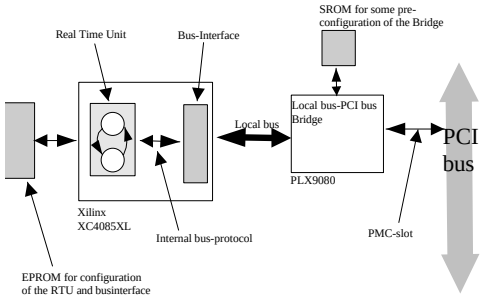

Figure 4: RTU-PMC block diagram

In the SARA-system, the Real Time Unit is attached to the local PCI bus on the board

in the system slot. When the RTU signals task-switch, it will generate an interrupt to the CPU that will perform the task-switch. As the RTU is attached to the local PCI-bus on the system-board, all signals will appear on the backplane as well (through the transparent PCI-PCI bridge). Interrupts are normally generated through the four interrupt- lines (INTA-D) that are available on the backplane. However, if more than four boards are inserted, some boards must share interrupts which may cause latencies.
The solution to this, in the SARA-system, is to use a 'doorbell' register. This register is implemented in the non-transparent bridge. It is 16 bits wide and when any bit is set in the register, it will cause an interrupt to its local processor. When the RTU wants to signal task switch, it generates a write-cycle from the local PCI bus where it is hosted, through the PCI-PCI bridge [DEC21554 HW Ref Man] to the CPCI-bus and the doorbell-register it wants to access. All doorbell registers have a unique address. By using these doorbell registers, the problem of shared interrupts is avoided. Another advantage is that it is possible to send 2^{16} different interrupts. Interrupts to the system-board are generated by INTA. As this board is the only user of this interrupt, there are no problems with latency. The only latency that has to be considered, when the doorbell registers are used, is the time it will take to perform the write-cycle.

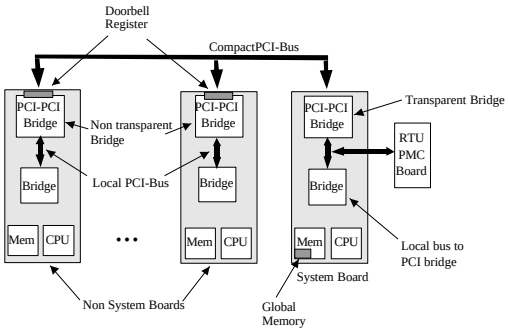

Figure5: Block diagram of SARA-System

Conclusions

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

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. The hardware support makes the IPC protocol predictable and gives short response time. Today some parts is still implemented in software, but in the next version it will be in hardware which makes the response time and time gap between the best and worst case smaller. Also standard components with other interfaces than IPC, can be instantiated with an adapter between the slot and the new component. We will also try to integrate old systems with the SARA concept, for example a conventional real-time operating system.

When one critical function is implemented into a hardware unit, the response time and time gap between the best and worst execution time decrease. 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.

Today programming languages for real-time applications must also provide the flexibility to express various timing requirements. Database and artificial intelligence are systems that probably will be more common in complex systems.

Future expansions and works:

- Expand SARA concept to hierarchical processor systems (more than 20 processors).
- Cache analyze, today we use two level caches. Some parts of the system can be using cash consistent protocol and others cannot.
- Debugging tools for multiprocessor system (hardware/software probes, tracing, logging etc).

- Change the scheduling algorithm to take into account busloads and cost of task switch.
- Prefetching mechanisms for task start and communication.
- Computer models of the system, to analyze behavioral of different situations.
- Analyze tools, to help the system designer to tune in the system for best utilization.
- Different software/hardware verification tools.
- A new scheduling algorithm for critical tasks.

Acknowledgements
These projects are sponsored by KK-Foundation, internal founding by Mälardalen's University College, ABB Robotics (Peter Ericsson) and Ericsson Utvecklings AB (Lars Ternsjö)

References

[Lawson98] Harold W. Lawson, "Salvation from System Complexity," Computer, Vol. 31, No. 2, February, 1998, pp. 118-120.

[Lindh95] 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

[Molesky90] L. D. Molesky, K. Ramamritham, C. Shen, J. A. Stankovic, G. Zlokapa, ”Implementing a Predictable Real-Time Multiprocessor Kernel - The Spring Kernel”, 1990

[Stancovic] 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.

[Kopertz91] 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 System Design

[98Mohammed] M. El Shobaki, "Verification of Embedded Real-Time Systems Using Hardware/Software Co-simulation", In Proceeding Vol I of the 24th Euromicro Conference pp. 46-50, Västerås, Sweden, August 1998. [Dal94] M. Dal Cin, W. Hohl,.... Architecture and Realization of Modular Expandable Multiprocessor System MEMSY, First Intl. Conf on Massively Parallel Computing Systems (MPCS’94), Ischia, Maj 2-6, IEEE 1994, pp. 7-15, 1994

[Furunäs97] J. Furunäs, J. Adomat, L. Lindh, J. Stärner, P. Vörös,”A Prototype for Interprocess Communication Support, in Hardware”, 9th Euromicro Workshop on Real-Time Systems, 11 - 13 June, 1997, Toledo, Spain.

[Stärner96] Real-Time Scheduler in Hardware, Johan Stärner, Johakim Adomat, Johan Furunäs och Lennart Lindh, Euromicro Conf'96 Prague.

[PICMG97] PCI Industrial Computers Manufacturers Group CompactPCI Specification Revision 2.1

[DEC21554 HW Ref Man] Digital Semiconductor 21554 PCI-to-PCI Bridge for Embedded Applications. Hardware Reference Manual.

[Hennessy95] John L. Hennessy and David A. Patterson , Computer Architecture: A Quantitative Approach, Second Edition 1995, ISBN 1-55860-329-8