Adding Flexibility and Real-Time Performance by Adapting a Single Processor Industrial Application to a Multiprocessor Platform

Leif Enblom, Lennart Lindh
Mälardalens University, MRTC (Mälardalens Real Time Research Center)
{leif.enblom, lennart.lindh}@mdh.se

Abstract— *This paper describes a way to get more flexibility in a real-time product and its base platform (real-time operating system and hardware). Industrial hardware and software platforms are due to change and in some cases a new platform is needed after five to ten years, if not earlier. This is costly and there is a need to be able to make the product grow in performance without changing the platform. The ongoing work that is described in this paper is performed in cooperation with industry and the attempt is to convert a single processor software application to a multiprocessor application. By changing the platform to a flexible multiprocessor real-time platform, flexibility and performance will be increased, resulting in a more optimized platform for different configurations of the application.*

Index Terms – multiprocessor system, scalable multiprocessor system, real-time system

I. INTRODUCTION AND MOTIVATION

A multiprocessor application in an industrial system is often different compared to a massively parallel application. Usually a small number of processors collaborate and applications do not act on very much data. The challenge is rather concerning response time, where a late response is an error and can result in damage to equipment or even humans. Response time constraints range from a couple of ms up to around 20ms for the application described in this paper, depending on where in the system measurement is performed.

The need for more real-time performance in many industry control and protection systems is increasing from year to year. New functionality is added into systems, which may alter system behavior, timing constraints and overall performance. A simple and maybe not very costly temporal solution is to follow the evolution of ever-increasing performance of CPUs, upgrading the system when calculation demand exceeds maximum calculation power. This may not be attractive for industrial systems for a number of reasons. Industrial systems, that are located in harsh electrically and magnetically influenced environments, may encounter limited possibilities for cooling and protecting equipment. Sometimes cooling and protecting is not possible or even forbidden. Thus, increasing computing performance by increasing the clock-frequency on the board is not in itself feasible, if this implies more power consumed. Multiple boards on the other hand may require as much or even more power than a system with one CPU, but cooling is made easier. For some environments, requirements are restricting power consumption for a single board to a specific amount, while the allowed power consumption in the whole rack-mount, where the boards are placed, is greater. In these environments it is attractive with a flexible solution where the application designer can choose to use one or multiple boards.

The work that is progressing, attempts to evaluate the possibilities to create a system that is not limited to a specific number of CPUs. Both the hardware and software system should be scalable, and the system today is for the main part based on standard components, which should be the case for the new system as well (with exception of the RTU[2]). The use of standard components is attractive and can give faster time to market as well as cutting development costs. Hardware designers are hard to find and standard components are well tested and comparably cheap. The design of a new PCB (Printed Circuit Board) with a high clock-frequency is not easy to implement, and a standard PCB can reduce these problems.

An interesting aspect of the adaptation to a multiprocessor system is the flexibility of hardware configuration. Sometimes the calculating demand of the system used today exceeds the possibilities to increase performance on a single board, and an identical system (in another rack-mount) has to be built to support all additional functionality needed. This identical system is inserted in parallel, communicating mostly over fiber-network interconnect, thus increasing the hardware needed. In this case the proposed multiprocessor solution can decrease the number of components used, limiting the need for an additional rack-mount.

II. THE SOFTWARE MODEL AND THE HARDWARE ARCHITECTURE

The SARA-system (Scalable Architecture for Real-time Applications)[1] is a system based on the idea to incorporate as many parts of a real-time operating system into hardware as possible. In this system the central part is the hardware RTU (Real Time Unit)[2] that is responsible for many operations commonly carried out in software. The scheduler is for example integrated into the RTU, capable of controlling tasks on a variable number of CPUs in the system. The scalability of the SARA-system can be used in the transition from a single processor system into a multiprocessor system. In this work the SARA-system is inspirational in converting the original system into the new.

An overview of the original hardware architecture can be seen in Figure 1. The present system is a single processor

system, mounted in a rack-mount with a standard back-plane Compact PCI-bus. Asynchronous data, originating mainly from the AIM (Asynchronous Input Module), stream into the system towards the CPU-board. The system has to evaluate the data continuously, and make appropriate decisions on time. There are strict criteria on the response-times to events that have to trigger the actuator.

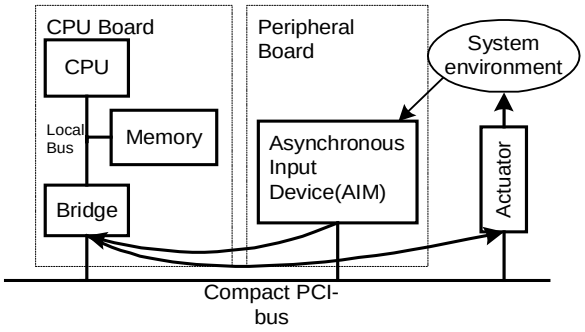

Figure 1, The original Hardware Architecture

The software model of the original system is depicted in Figure 2, with applications utilizing the system by communicating with a VOS (Virtual Operating System). T1 – T4 in the figure depicts tasks in the application and base-system executing on the CPU-board. The main reason why VOS was created was portability, enabling the developer to move existing applications from one platform to another without having to adopt the code to a new operating system and hardware. Different real-time operating systems can be run below the VOS interface, for example WindRiver's VxWorks[3] or any real-time operating kernel supporting the features of VOS.

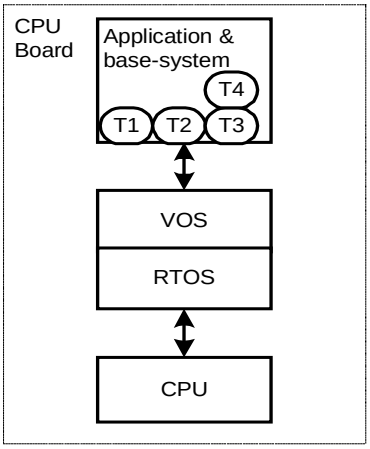

Figure 2, The Software Model of the Original System

The new proposed hardware architecture looks as depicted in Figure 3. As can be seen, the RTU and an additional board have been introduced where the RTU schedules tasks running on both CPU Board 1 and CPU Board 2. In this MIMD[4] architecture, tasks being scheduled by the RTU run independently on the boards. Independently means that the code that executes does not need to be aware of on which board it runs. There is no need for a scheduler on each of the boards, thus reducing time spent in context switches. As many parts of the system as possible consist of standard components, including the boards, the rack-mount and the back-plane bus, but there still exist some legacy in-house components (such as the AIM), and the RTU is a custom defined component.

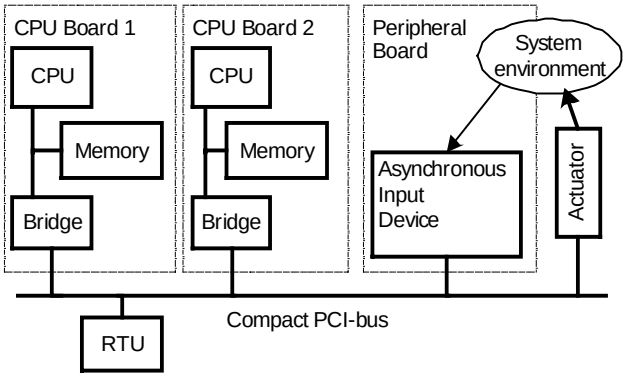

Figure 3, The new Hardware Architecture

The software model of the system in the new multiprocessor system is illustrated in Figure 4. Context switches are triggered by the RTU, and the RTOS on each board saves current context and makes next task running.

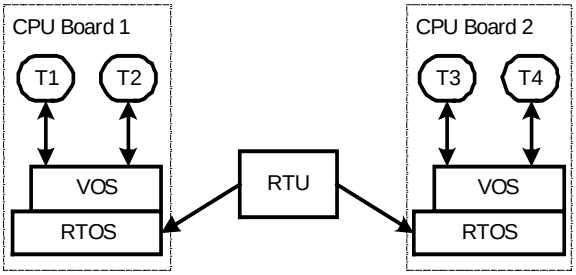

Figure 4, The Software Model of the New System

The RTOS provides an interface for accessing the features of the RTU, changing priority of the tasks, setting periodic timers and interrupt-handling methods.

III. SOFTWARE CONSIDERATIONS

There are different aspects of the software system that has to be evaluated and analyzed. Today at least three areas of interest can be outlined as described below.

A. VOS

VOS is the main interface between existing applications built for the system. A number of fundamental calls and functions of the VOS have to make use of the RTU in the new system, and among the features that have to be supported are:

1. Threads have to be able to be deleted and created, as well as being suspended. When the task is to be run is decided by the RTU. This is done by issuing an interrupt to

the CPU-board where the task resides, triggering the local task-switch routine.

2. Shared resources have to be protected by binary semaphores and counting semaphores. Resources that have to be protected are of special interest in a multiprocessor system. There has to be a central resource so that the different CPU-boards can take and release the semaphores atomically. This is possible through a hardware semaphore implemented in the RTU.

3. Memory handling is essential in the system, where tasks in the original system allocate memory from pre-configured memory-pools. This feature will function alike in the new system.

4. Message passing between tasks in the original system is performed through message queues that can be globally accessed by every task. Messages can be sorted by priority as well as handled in FIFO order. In the new multiprocessor system, message passing can be integrated into the RTU by means of messages queues in hardware. A reader wanting to receive messages from a message queue gets blocked when there are no available messages, and whenever a message arrives at the queue the task get ready to run, i.e. it is put into the ready-queue of the RTU.

5. Periodic triggers as well as delay functionality are essential in a real-time system and are supported by the original system and in the new as well.

B. I/O and Interrupts

Interrupts are an essential issue in the present system. A frequent amount of incoming data needs to be handled by the application, originating from I/O boards (mostly the AIM). Handlers, i.e. callback functions, take care of incoming signals on the single CPU main board. A uniform handling concerning interrupts is needed in the new system and tasks have to be partitioned onto the boards in a right way. It is desirable to allocate all ISR (Interrupt Service Routines) on one single board thus collecting the base-system there. Applications can then be placed on the secondary board, relieving the pressure on the base-system, and simultaneously the application gets more computing time by not being interrupted by the base-system. This will be more discussed in chapter four below.

C. Applications

Applications written for a single processor system can inherently have semantics and code that pose a threat to an adaptation to a multiprocessor platform. Message passing is for example performed in the original system through message buffers that reside solely on one board. The new system must support uniform use of the send and receive primitives across the whole platform, and this is accomplished through the use of message queues integrated in the RTU. This gives the possibility to write and adapt application message passing that scales arbitrarily in the multiprocessor system.

A uniform approach to the use of shared resources has to be developed, which includes for example semaphores. Semaphores that can be taken locally in the original system have to conform in the same way in the multiprocessor system. The RTU has got support for hardware semaphores that can be used to achieve a centralized access-point to semaphores and protect resources.

IV. Flexibility and Real-Time Performance

The challenging part of the work is the adaptation of the application, which has solely been written for a single-processor system and utilizes a single processor RTOS. Questions arise concerning the tasks and on which boards they are to be placed to achieve an optimized application. There is a need in the system to differentiate the real-time parts of the system (generic I/O handlers and real-time tasks having timing constraints) from less timing constrained tasks such as logging functions and external HMI (Human Machine Interface) communication. Different configurations for where to place tasks on the different boards need to be developed. Users of the system must be able to easily alter the configuration and test it on the target system. The solution is to provide a configuration mechanism enabling the developer to move and compile tasks into different images that can be downloaded to the boards.

Communication between the boards has to be minimized to gain best performance in the multiprocessor system. The original application has been created and partitioned into many tasks as illustrated in Figure 5 (the tasks merely illustrate how the application can be configured). Different groups of tasks that share common assignment and cooperate a lot can be identified in the original system. These tasks should be placed on the same board, and the developer of the system can with a configuration manager place them arbitrarily on the different boards.

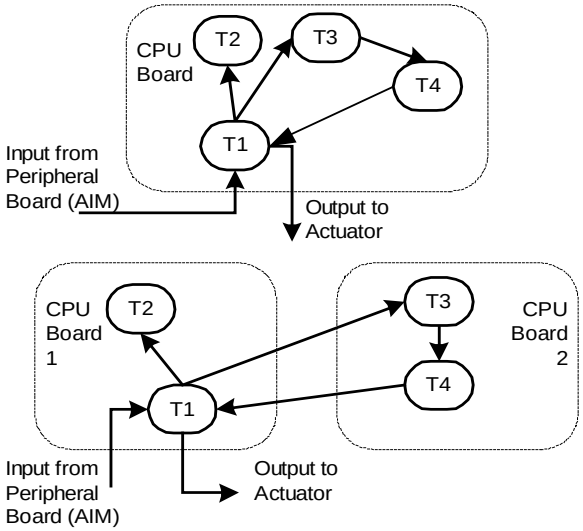

Figure 5, example of tasks in the original system as well as the new multiprocessor system and a possible configuration.

Task T3 and task T4 in Figure 5 may for example cooperate to a high degree, while task T1 is receiving a high volume of data from the peripheral board (AIM). The data is prepared for the application (task T3 and T4) by task T1, and also sends some logging activities to T2. In finding a configuration where maximum performance is reached, tasks T3 and T4 might be placed on CPU board 2.

Data from the AIM is arriving at task T1 in bursts, and during these periods CPU 1 will be highly loaded. As task T1 is receiving data and preparing it, the application (task T3 and T4) can continue to run without interruption. Therefore a performance speedup in the application is possible. The application designer might want to separate task T1 from the application tasks as described, and this should be performed with the help of a partition manager at compile time.

Since we have introduced message passing between the boards, we also have introduced more delay into some of the data exchange. It is therefore especially important for the developer of applications to partition tasks well. The extra message passing between the two boards might potentially lead to bus contention and longer delays. First analytical results show that this is not a threat. The interconnect is able to deliver much higher volumes of data than is produced and sent.

A gain that is inherently provided by the RTU is a performance increase due to the lack of timer-interrupts on each board. The tasks do not require rescheduling or corresponding actions until the RTU issues a command. This gain can be as large as 32%[5], compared to a system with a conventional scheduler in software having to administer clock ticks.

Flexibility has been introduced in the system thanks to the ability to choose to build a single processor system or a multiprocessor system. It is not necessary to run the new system on a multiprocessor architecture, so for the cases where extra performance is not needed a single processor solution is sufficient. At the same time as we have introduced flexibility with the multiprocessor solution we have also introduced complexity in terms of increased difficulty debugging the application and more parameters to remember when writing applications. This requires extra attention on how the base-software and operating system is constructed. Application coders should not be forced to take special actions to send a message to a special board, for example.

V. FUTURE WORK AND SUMMARY

As the work progresses, knowledge of how a complete mapping of the system can be done is gained. While identifying the problems involved when integrating the multiprocessor system, problems may arise. These problems may arise in the synchronization of tasks, preserving the semantics of the applications, and providing full support for all the features required. Furthermore there is a need of identifying which additional features are needed in the RTU, and how they have to be developed to make the new system fully functional.

The SARA platform, which has been inspirational for this attempt, already today provides full multiprocessor support. Today's single processor applications in the present system are quite well parallelized due to wise use of threads, but performance is not sufficient for some configurations of the application. The multiprocessor solution utilizing a central hardware scheduler and resource manager (RTU) can increase flexibility in terms of scalability in hardware as well as software. There are big challenges involving adapting present code to the new system and the future will yield if the solution is commercially feasible in terms of development time and efforts.

VI. ACKNOWLEDGEMENTS

ABB Automation Products have supported this work.

VII. REFERENCES

[1] Lennart Lindh, Tommy Klevin, Johan Furunäs, "Scalable Architecture for Real-Time Applications - SARA", *Swedish National Real-Time Conference SNART'99* Linköping, Sweden, August 1999.

[2] Joakim Adomat, Johan Furunäs, Lennart Lindh, Johan Stärner, "Real-Time Kernel in Hardware RTU: A step towards deterministic and high performance real-time systems", *8th Euromicro Workshop on Real-Time Systems*, L´Aquila, Italy, 1996.

[3] http://www.windriver.com

[4] Michael Flynn, Kevin Rudd, "Parallel Architectures", *ACM Computing Surveys*, Vol. 28, No. 1, March 1996.

[5] Johan Furunäs, "Benchmarking of a Real-Time System that utilizes a Booster", *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA2000)*, Las Vegas, USA, June 2000.