Hardware Accelerator for Single and Multiprocessor Real-Time Operating Systems

Lennart Lindh, Johan Stärner, Johan Furunäs, Joakim Adomat and
Mohamed El Shobaki
llh@mdh.se, jsr@mdh.se, jfs@mdh.se, jat@mdh.se och mei@mdh.se.
Mälardalens högskola, Sweden

Extended Abstract

Real time kernels are implemented today in software or in a separate processor. One of the disadvantages of current implementation approaches is that the execution times for the Service Calls have a minimum and a maximum value. The time gap can be large, and in real time systems the worst case time is one of the factors which determine the utilisation factor of the system. The execution time variations are dependent on the longest and shortest execution paths in the source code, the execution time for the assembler instructions and load variations such as number of tasks, interrupts, time-out queues etc.
The real time Kernel in hardware is designated **R**eal **T**ime **U**nit (**RTU**). The RTU is designed to reduce the time gap between best and worst cases for the kernel's service calls and administration to zero. Real time administration such as scheduling of tasks and time queue handling is performed outside the CPU and therefore no clock tick administration routine interrupts executes in the CPU. The CPU task is to perform program execution and context switches, in today's implementation.
As distinct from other methods of implementing real time systems (for example [Roos89], [Stancovic95]), the entire real time kernel is implemented in a number of separate real ”executing” hardware units. This means that there are no microcode [NEC92] or ROM-based operating system instructions such as TRON [Sakamura89].

The left hands part of figure 1 shows a conventional real-time system with the operating system in software. It consists of three parts; Application software, Operating system and Processors. When using the RTU, some part of the real-time operating system is moved from software into hardware, as shown in the right hand part of figure 1.
Predictability [Stankovic90] of the underlying real-time operating system is necessary to achieve predictability of software (application tasks) running on top of this operating system. One approach to improving predictability is to use a separate real-time coprocessor.

The validation system includes an RTU chip, VME bus [VME] and three CPU cards from FORCE [FORCE] (see section architecture). The RTU-project has designed 10 different FPGA and three ASIC RTU prototypes the last years [Lindh95].

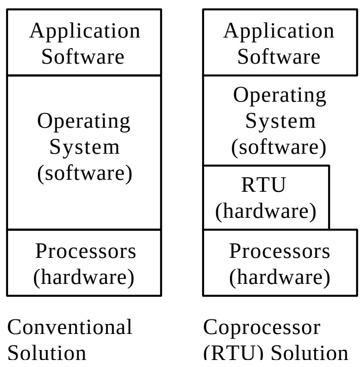

Figure 1: Overview of software and hardware implemented Real-Time Kernels

The RTU prototype can handle 256 tasks and 32 priority levels. The RTU is designed as a framework, for the convenient implementation of "new" concurrent service calls.
The following real time functions are implemented in the RTU: Interrupt handler, periodic starts of tasks with deadline control, relative delay of tasks, scheduler (priority based, pre-emptive), IPC, semaphores, flags, watchdogs, analyse and debug functions, on/off task switch and VME bus interface.
The RTU implementation is technology independent [Lindh93] and it can be implemented as a megacell, one Integrated circuit (100 000 gates) or in FPGA:s (**F**ield **P**rogrammable **G**ate **A**rray).

What is the difference between a system with an RTU and the real-time kernel in software?
A "true" definition of the advantages of the RTU is not possible in a general way, because the requirements and constraints are very different for different applications.
For example, a robot requires a real time system with a response time in milliseconds whereas the real time system of a water supply plant needs only a response time of the order of one second.
Given the qualification concerning applications, we observe that the use of a separate hardware-based real time kernel yields the following advantages:

PERFORMANCE AND DETERMINISM

- Possible to get a more deterministic multitasking real time kernel, since RTU service calls min. max. time is zero.
- The speed with which real time service calls are executed is increased by several orders of magnitude compared with a conventional CPU based solution.
- The response time has been decreased by several orders of magnitude in comparison with a CPU solution,
- No clock tick interrupt to the CPU is needed.
- ”Execution” of expansive software based function can be implemented in hardware.

PORTABILITY

- No memory requirements,
- No special instruction format,
- No special CPU, only one interrupt and an external data bus for each CPU.

SOFTWARE DESIGN

- Simpler software system, since the program code for the real time kernel does not occupy CPU memory,
- Simpler timing analysis of the system,
- Improved understandability when the system is divided into parts (complexity reduction),
- All RTK service calls are isolated in the RTU (no problem with interference between different service call codes),
- No interrupt handler in the software (except the context switch routine) since interrupts are mapped to tasks, which are scheduled by the RTU.

In principle, the motivation for using an RTU is the same as that for using a coprocessor in a computer system, i.e. to improve the performance of the application. A math-coprocessor is specialised in performing fast calculations and an RTU is specialised in high performing and predictable time behaviour for the real time kernel.

System Architecture and Interface to RTU

The figure below shows our validation environment. The number of application processors controlled by the RTU is scaleable from one to three. Each CPU board has I/O circuits and RAM (Random Access Memory). The global RAM module can be replaced with a reflective memory thereby reducing bus traffic. The bus arbitrator controls accesses to the bus. The CPUs as well as the bus, memory, arbitrator, and I/Os are standard off-the-shelf components.

The interface to the RTU consist of a number of registers, some shared by all the CPUs, and some CPU specific. The common registers are used to define system specific behaviour. Each CPU communicates with the RTU through a set of control, status, and service request registers. Refer to [rtu96fnc] for further details.

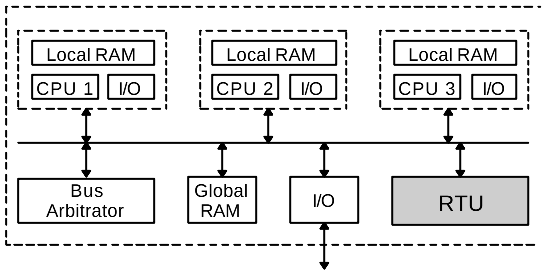

Figure 2: System Architecture

The ability to migrate task between the CPUs requires that some part of the memory is assessable by all CPUs. This reflective, or global, memory stores common data structures such as stacks and task control blocks. The task code and data structures fixed to one CPU are stored in the local memory.

The RTU operates synchronously with the CPUs, and therefore an acknowledge-based protocol is required. The communication protocol is supervised by an optional time-out, enabling fast detection of errors in the handshake protocol.

The RTU uses an interrupt to notify the CPU that a task switch is to take place. The task switch request is acknowledged by the CPU, and the currently running task status is stored in the TCB (Task Control Block). The next running task is then fetched from the RTU, and its status is loaded from the TCB into the CPU registers.

References

[FORCE] Manual SYS68K/CPU-1,FORCE Computers Advanced Systems, FORCE Computers GMBH, Daimlers 9, 8000 München, Germany. ASICs, pages 449-454, Microprocessing and Microprogramming (24), 1988.

[Lindh93] L. Lindh, Prof K.D. Müller-Glaser and H. Rauch, Rapid Prototyping With VHDL and FPGAs, Lecture notes in Computer Science 705, Springer-Verlag, ISBN 0-387-57091-8 or ISBN 3-54057091-8, 1993.

[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,

[NEC92] K. Bulla and R. Turski, "Echtzeit im Auto"; Seiten 66 - 68; Design & Elektronik; Nr. 2 vom 3.11.1992; Markt & Technik.

[Roos89] Jochim Roos, The Design of a Real-Time Coprocessor for Ada Tasking, pages 17.0-17.12, NORSILC/NORCHIP Seminar 1989, Stockholm, Sweden.

[rtu96fnc]J. Adomat, J. Furunäs, J. Stärner, L. Lindh ”RTU96 Functional Specification”, Internal, Mälardalen University, P.O. Box 883, S-721 23 Västerås, Sweden, 1995.

[Sakamura89] Ken Sakamura (ed), TRON Project, 1989, Springer Verlag, ISBN 0-387-70050-1, 1989.

[Stankovic95] J. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo, Implications of Classical Scheduling Results For Real-Time Systems, IEEE Computer, Vol. 28, No. 6, pp. 16-25, June 1995.