FASTHARD - A Fast Time Deterministic HARDware Based Real-Time Kernel

Lennart Lindh
University of Eskilstuna/Västerås
Department of Real Time Computer Systems
P.O Box 11, S-721 03 Västerås, Sweden
Tel xx46 21 101300
Fax xx46 21 142017

Abstract

Real-time systems handle increasing complexities of functions and require better determinism and higher speed. One way to make it easier to meet those requirements is to implement the real-time kernel in a separate unit working concurrently with the CPU, called RTU (Real-Time Unit).

This paper describes an RTU we call FASTHARD as a model in VHDL (hardware description language). It is planned to be implemented in a special hardware architecture. All standard CPU’s can be used together with FASTHARD. FASTHARD can handle rendezvous, interrupts, periodic start of tasks, delay tasks and activate/terminate tasks without any contact with the CPU.

1 Introduction

In many of today’s real-time systems the CPU has to execute both the RTK (Real-Time Kernel) and the task code ([Transputer88] and [Timno]). In such systems the RTK decreases the execution time available in the CPU for the tasks. Another problem is that the execution time for the RTK often is a function of many variables such as number of tasks, clocktickperiod, task scheduling algorithm and communication protocol. For example, the real-time kernel VRTX needs 144 μs execution time from the CPU (Motorola 68000) to create a task, about 100 μs for task switching and the clocktick interrupts require (14+7*n) μs, where n is the number of tasks [Timno82]. This variable execution time for the RTK often leads to non-determinism for the real-time system.

One solution of the problem is to implement the real-time kernel in a separate hardware unit. We define this unit as RTU (Real Time Unit, [Stanischewski&Lindh91]).

We get the following benefits in using a separate hardware unit for the Real-Time kernel:

• the CPU has nearly 100% execution time for the tasks,
• simple software system, because the program code for the Real-Time kernel is not needed in the CPU memory,
• easier to debug the real-time software, because interrupts and different protection modes are not needed (only one interrupt require),
• possible to build a deterministic system (the theory of real-time scheduling require deterministic Real-Time kernel),
• possible to have complex task scheduling algorithm (the scheduling algorithm is implemented in FASTHARD and doesn’t take any execution time from the CPU).

Instead of the traditional ways to implement a Real-Time kernel ([Transputer88], [Tino82], [Roos89] and [Juntunen88]), RTU contains the whole real-time kernel in specialized hardware. That means there are no micro-coded or ROM-based operating system instructions like in TRON [Sakamura89], Transputer [Transputer88] or EPROM [Tino82]. The difference between FASTHARD [Stanischewski&Lindh91] and FASTHARD is that a standard processor is used instead of a specialized processor.

2 Overview

FASTHARD can handle 256 tasks and 8 priorities. From figure 1 man can infer that we have followed real-time service calls for each task in FASTHARD.

• Activate Task: activates another task.
• Terminate Task: terminates itself, afterwards it can only be activated by another task.
• Delay Task: deactivates task for a constant time.
In FASTHARD there is a RTK implemented as described in figure 1. This runs automatically and concurrently to the CPU.

A simple Real-Time system (see figure 2) consist of a CPU, main memory, I/O ports and FASTHARD (RTU). The system bus and an interrupt line connect CPU and FASTHARD. Eight external interrupts are connected to FASTHARD.

3 Functionality, performance and determinism in FASTHARD

A brief functionality description of the SVC (Service Calls) handled by FASTHARD follows in C:

- RELATIVE_DELAY(time); RETURN(); (suspends the executing task for given time)
- ACTIVATE(task_id, priority, start_address);
  RETURN(); (activates task )
- TERMINATE(); (terminates the running task)
- INIT_PERIOD_TIME(period_time); RETURN();
  (initiates the period time for the executing task)
- WAIT_FOR_NEXT_PERIOD();
  RETURN_TSW(missing_deadlines); (task waiting for a periodic start)
- OFF_PERIOD_START();
  RETURN(return_already_off); (no periodic starts for the executing task)
- CALL(entry, msg_pointer,task_id, time_out)
  RETURN_TSW(time_out); (calls an entry of a task)
- ACCEPT(entry, msg_pointer,time_out);

RETURN_TSW(time_out, call_task_id)
  (accepts a call to the specified entry)
- COMPLETE(); (completes a rendezvous)
- OFF_TASK_SWITCH();
  RETURN(ALREADY_OFF); (no taskswitch allowed)
- ON_TASK_SWITCH(); RETURN(ALREADY_ON);
  (taskswitch allowed)
- WAIT_IRQ_EXTERNAL(irq_nr, time_out_time);
  (task waiting for an interrupt)

Eight external interrupts are provided, the priority level of the external interrupts are the same as the waiting tasks priority. If an external interrupt occurs the task is moved to ready queue.

FASTHARD function model (see figure 3) has similarity to other Real-Time kernels (pSOS-68k, O'Tool).
Example queues, timeout functions and scheduling algorithms are the same as in other RTK's. The difference is

<table>
<thead>
<tr>
<th>ADR</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>HS_TSW</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NEXT_TASK_ID</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>BLOCK_TSW</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>CALL_SVC</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HS_SVC</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>RETURN_DATA</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>PARAMETER_DATA</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4: Register select bits  address bit 2-0 and R/W-N bit
the implementation. The commercial RTK is implemented in software and FASTHARD is implemented in about 60 concurrent statemachines.

Most SVCs are handled in less than 10 assembler instructions. Tist includes the overhead of blocking for task switch, error check and parameter transfer to and from FASTHARD.

4 FASTHARD Interface

The FASTHARD contains seven registers and one interrupt line to the CPU. The registers are: HS_TSW (HandShake_TaskSWith), NEXT_TASK_ID, BLOCK_TSW, CALL_SVC, HS_SVC, RETURN_DATA and PARAMETER_DATA register.

IRQ_CPU, HS_TSW and NEXT_TASK_ID are task switch handling registers and interrupt line. The register HS_TSW is a handshake (HS) register, bit 0 in the register is the handshake signal, the others don't care (bit 15-1). After an interrupt (IRQ_CPU) the task switch interrupt routine starts and HS_TSW sets to one. After having saved old CPU registers to TCB (Task Control Block), the address to the new registers are fetch from the NEXT_TASK_ID register. Next task registers load to CPU. The last activity to change task is to load the program counter in the CPU with the new tasks program counter (from TCB).
The interrupt routine to the CPU is briefly like this:
SET "HS_TSW"; acknowledge interruption.
SAVE old registers to TCB (Task Control Block)
LOAD the next task register from the TCB.
RESET "HS_TSW";
JMP (NEXT_TASK_ID_PC); (Program Counter)

BLOCK_TSW, CALL_SVC, HS_SVC, RETURN_DATA and PARAMETER_DATA are used for the SVC. The CALL_SVC register is a control register for the SVCs (see figure 5). It is important to block the task switch before parameters are sent to FASTHARD (BLOCK_TSW = 1), because of the resource sharing of the register in the FASTHARD.

HS_SVC is the status register and bit 0 is one when FASTHARD has accepted the SVC call.
RETURN_DATA is a 16 bits word and PARAMETER_DATA contains 16 bits data.

Briefly the code for a SVC call is structured like this:

SET "BLOCK_TSW"; (No taskswitch allowed)
LOAD "PARAMETER_DATA"; (Loads parameters to the RTU)
SET_BIT "CALL_SVC";
WAIT UNTIL "HS_SVC" = 1; (Handshake signal)
LOAD return_data WITH "RETURN_DATA"; (Returns data to the task)
RESET_BIT "CALL_SVC"; (Task is ready)
WAIT UNTIL "HS_SVC" = 0; (FASTHARD is ready)
RESET "BLOCK_TSW"; (Taskswitch allowed)

TCB (Task Control Block) contains register data (when it’s not running), start address for the task and message data or pointer for rendezvous.

5 Conclusion

The paper has showed that an useful real-time-kernel can be implemented in special hardware with better performance and determinism than in software or microcode. Our first model is a simulation model (subset of VHDL) in a logic synthesis language. In the summer of 1992 we plan to have a prototype FASTHARD in hardware.
The further work will be to expand the FASTHARD VHDL model with new functions like deterministic and high performance Real-Time communications between CPU's.
References

[Tinno82] John Tinno, Real-Time Operating System
Puts the Execution on Silicon, pages 137-140,
April 21, 1982.

[Motorola 68000] Motorola Inc. 16, chemin de la Voie-
Creuse, P.O. Box 8, 1211 Gene ve 20, Switzerland.

[Transputer88] The Transputer Databook, INMOS,
1988

[Juntunen88] T. Juntunen, J. Kivelä, A. Reinikka, M.
Sipola, J.-P. Soininen, K. Tiensyrja, T. Tikkanen,
Real-Time Structured Analysis in System Level
design of Embedded ASICs, pages 449-454, Mi-
croprocessing and Microprogramming (24), 1988.

[Sakamura89] Ken Sakamura (ed), TRON Project

[Stanischewski&Lindh91] Lennart Lindh, Frank
Stanischewski, FASTCHART - A Fast Time De-
terministic CPU and Hardware Based Real-Time-
Kernel, IEEE, Euromicro workshop on Real-Time
Systems, 1991. And, FASTCHART - Idea and
Implementation, IEEE International Conference
on Computer Design (ICCD), Boston, USA (14-16

[pSOS-68k] IRONICS Incorporated, Computer Systems
Division, 798 Cascadilla Street, Ithaca. New York
14850, USA

STEMS AB, Sal dov 9B, S-175 62 Järfäll, Sweden