Emulation of a Fast Reactive Embedded System using a Real Time Operating System

Karlheinz Weiß, Thorsten Steckstor, Prof. Dr. Wolfgang Rosenstiel

University of Tübingen, Technical Computer Science, Sand 13, D-72076 Tübingen, Germany

weiss@fzi.de, stecki@fzi.de, rosen@informatik.uni-tuebingen.de

Abstract

This paper presents the emulation of an embedded system with hard real-time constraints and response times of about 220µs. We show that for such fast reactive systems, the software overhead of a real time operating system becomes a limiting factor. We analyze the influence of novel microcontroller features, e.g., different on-chip caches, which tend to accelerate execution, but make it less predictable. These investigations have been conducted using our own emulation environment called SPYDER-CORE-P1.

1 Introduction

In the last few years, the rapid progress in microelectronic technology has reduced component costs while simultaneously making the introduction of 32 bit embedded microcontrollers in a widespread area of embedded system design[1] quicker. Additionally, powerful on-chip features like data and instruction caches, programmable bus interfaces and higher clock frequencies provide an enormous performance speed-up and simplify system design. These hardware fundamentals enable the implementation of a real-time operating systems, which lead to the rapid increase in total system performance and the complexity of the functionality.

Nevertheless, if fast reaction times of about 220µs must be guaranteed, the software overhead due to task switching becomes a limiting performance factor and the effect of the caches makes the performance analysis very difficult. These issues will be addressed in this paper.

2 Emulation platform SPYDER-CORE-P1

The SPYDER-CORE-P1 environment provides all the key components needed for embedded systems in the area of industrial automation within a flexible, but still compact architecture, which guarantees high clock speeds. This enables a real-time emulation, which is very close to the final target system. The architecture is shown in figure 1.

A 32 bit RISC embedded PowerPC 403GA/GCX microcontroller is used. The high-speed microcontroller bus is connected to the main memory and the interface FPGA. It can be used to connect the entire microcontroller bus to external devices via the extension header 2 or it can be used to emulate additional application specific interface hardware.

A driver device implements a buffer between the high-speed 32 bit wide microcontroller bus and the slower 8 bit I/O bus. It connects some frequently used communication and memory devices to the microcontroller. The extension header 1 is used to connect debug equipment closely to the microcontroller (e.g. a logic analyzer).

[1] This work was supported in part with funds of the Deutsche Forschungsgemeinschaft under reference number 3221040 within the priority program "Design and design methodology of embedded systems" and by BMBF project 01 M 3035 A.

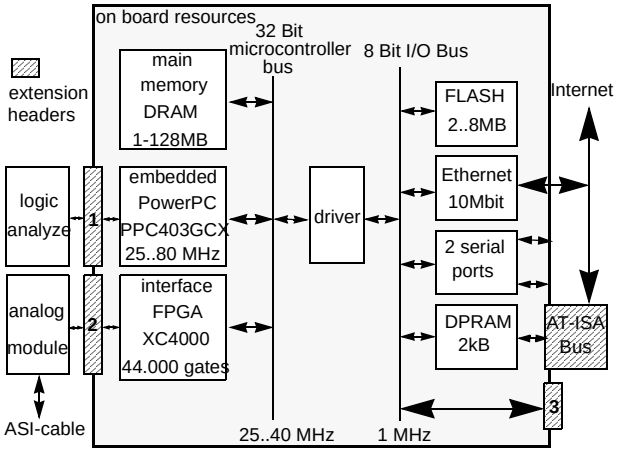

Figure 1. Architecture of SPYDER-CORE-P1

3 ASI-Master Application Benchmark

The Actuator Sensor Interface *(ASI)* connects up to 32 *ASI* slave chips via a single, bifilar cable with a *ASI* master unit

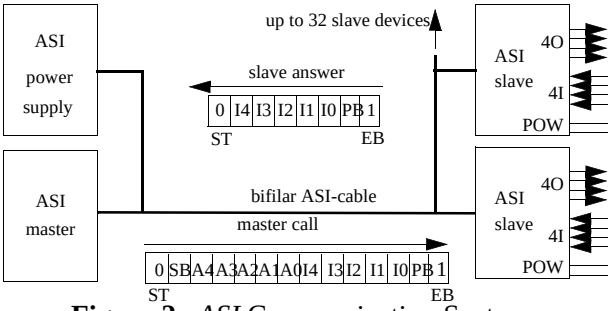

Figure 2. *ASI* Communication System

That master unit calls in a polling cycle each connected *ASI* slave with its own address (bits A4..A0), followed by the output data image (bits I4..I0) as depicted in figure 2. The *ASI* slaves respond, if the address in the master call matches with their own address and transfer the input data image back to the master. The serial master call protocol must be modulated on the *ASI* cable using a dedicated analog module (see figure 1). The *ASI* master is emulated on SPYDER-CORE-P1 very close to the final target system without any major changes. The hardware connection between the microcontroller and the analog module (two wires, serial in and out) is implemented as a dedicated *ASI* hardware in the interface FPGA XC4000 (see figure 3). The microcontroller writes the output data image to the register file and the *ASI* UART performs a parallel to serial conversion with manchester encoding. The digital serial data output is modulated to the *ASI* cable by the analog module. The receive path operates analogously to the send path.

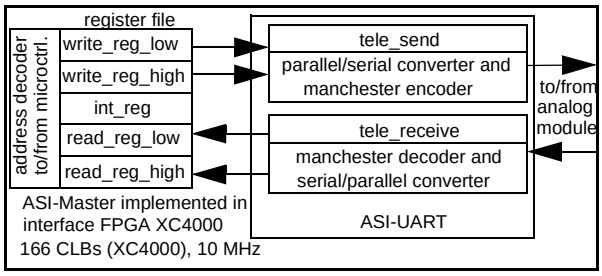

Figure 3. ASI Interface Hardware

The Real Time Operating System (RTOS) *VxWorks* runs on the SPYDER-CORE-P1. Four tasks run on the RTOS, two are hard real-time tasks and two tasks have no real-time constraints (see figure 4).

The *int_service* task is a hard real-time constraint task and is responsible for the data exchange with the slaves. It generates the current process data image. The *control* task is also a hard real-time constraint task and uses the current process data image to calculate the control commands.

The *server* task has no real-time requirements and is responsible for data and command exchange via the Internet. The embedded *http-server* task also has no real-time requirements and transfers JAVA applets to the calling client computer.

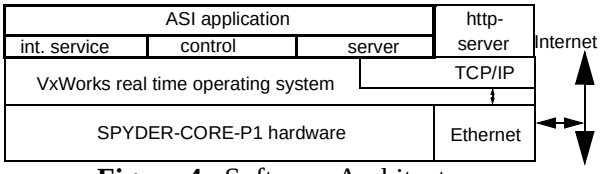

Figure 4. Software Architecture

4 Embedded System Performance Analysis

Figure 5 shows a timing diagram of the hard real-time tasks, measured with a logic analyzer. The int-signal shows the interrupt communication between the *ASI* specific hardware implemented in the interface FPGA and the microcontroller. The *int-service* and *control* tasks mark their begining and end states by accessing a simple I/O device (I/O-signal), which is recorded by the logic analyzer.

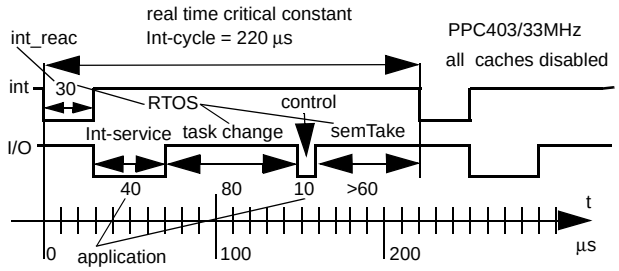

Figure 5. Execution Times

The *ASI* standard defines the time delay between to serial bits on the *ASI* cable during the master call and slave answer with 6µs. Together with master and slave break times, the microcontroller must exchange data with a slave every 220µs. Therefore, this value is an *ASI* specific real-time critical constant. During this time, the microcontroller must execute the following sequence: interrupt reaction, *int-service*, task change, run *control* task and take a semaphore (*semTake*). To determine the minimum clock speed, all caches must be disabled due to the fact, that this sequence is a hard real-time constraint and a worst case analysis has to be done. Figure 5 shows, if the worst case scenario occurs, it will still run successfully with a minimum clock speed of 33MHz or more. If the clock speed decreases, the next interruption (int = low) happens before the *semTake* currently being executed has finished. This means, the real-time requirement can no longer be satisfied.

Figure 6 shows the clock frequency range form 25 up to 80 MHz on the x-axis. The y-axis shows the ratio of the value of the real-time required for the execution of the mentioned task sequence and the real-time critical constant. The upper curve depicts the case without all on-chip caches, which must be used to determine the worst case scenario for hard real-time conditions. If the PPC403 operates at 33 MHz and the worst case scenario occurs (total cache misses and flushes), it is still able to guarantee the real-time requirement. If the clock frequency decreases, the y-value increases above one. That means, the microcontroller performance is undersized.

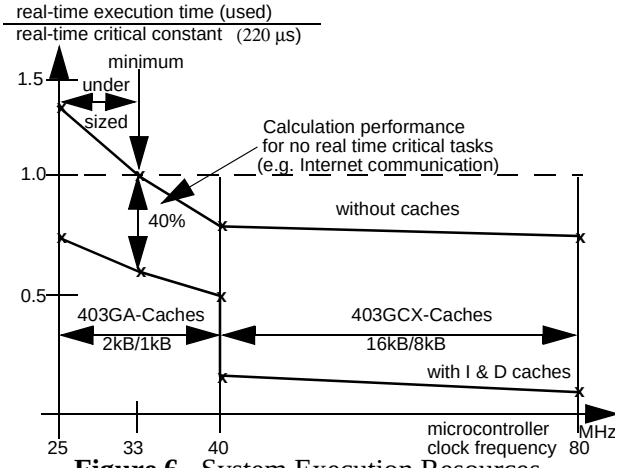

Figure 6. System Execution Resources

The lower curve shows the behavior with enabled D & I caches. This is the normal operation state. The minimal working frequency (33Mhz) provide an average gain of about 40% in execution performance. This 40% gain of the microcontroller execution performance can not be guaranteed in all cases, it is possible that lesser percentage occurs, although this performance gain usually can be expected. Therefore, these resources can only be used for no real-time tasks. In the range of 40MHz up to 80MHz, an further significant performance gain can be verified. This is due to the fact that the 403GCX type in that speed range used has eight times larger caches than the 403GA type.

5 Summary

The emulation gives a detailed view account of internal behavior in an early design stage and shows the minimal working frequency at 33MHz. The emulation reflects the final target system very closely without any change.

State of the art RTOS achieves task switching times of about 80µs and interrupt reaction times of 27µs. As shown in figure 5, the total time the RTOS uses is 170µs and the total time for the application is 50µs. Therefore, 77% of the total execution time (220µs) is consumed by the RTOS. That means, if the external environment of a fast reactive system forces interactions (here 220µs) to the same degree of magnitude as the task switching time (here 80µs), the software overhead of the RTOS becomes a limiting factor for the total embedded system performance.

Caches, which are enabled, improve execution performance and provide a gain of 40%. If a system requires hard real time conditions, this is done without cache enhancements. Such enhancements can only be used for non real-time dependent system services, e.g., network communication via the Internet.