Exploiting FPGA-Features during the Emulation of a Fast Reactive Embedded System

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

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

weiss@fzi.de, stecki@fzi.de, gkoch@synopsys.com, 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 (RTOS) becomes a limiting factor, consuming up to 77% of the total execution performance. We analyze features of different FPGA architectures in order to solve the system performance bottleneck. We show that moving functionality from software to hardware through exploiting the fine grained on-chip SRAM capability of the Xilinx XC4000 architecture, that feature eliminates the RTOS overhead by only a slight increase of about 28% of the used FPGA CLB resources. These investigations have been conducted using our own emulation environment called SPYDER-CORE-P1.

1 Introduction

Most of today's existing technical applications are controlled by so-called Embedded Systems. Many different application areas exist which demand their own specific embedded system architecture. Therefore, as described in [1], a common definition of embedded systems does not exist.

In the area of industrial automation, an embedded system architecture consists of an application specific hardware part, which interacts with the environment, and can efficiently implemented using a Field Programmable Gate Array (FPGA). At the same time, an application specific software part is running on a microcontroller. Especially the interaction with the system environment forces hard real time requirements.

In the last few years, the rapid progress in microelectronic technology has reduced component costs while simultaneously making the introduction of the 32 bit embedded microcontroller [2] in a widespread area of embedded system design[1] quicker. Additionally, novel FPGAs appeared in the market, which provide increased gate and routing capabilities. Their architectures are enhanced through powerful new on-chip features like SRAMs, flexible I/O buffers or fast carry logic. Smaller process technologies down to the deep submicron area for both microcontrollers and FPGAs achieve higher clock frequencies, which provide an enormous acceleration in performance and simplifies system design. These hardware fundamentals enable the implementation of Real Time Operating Systems (RTOS), which lead to the rapid increase in total system performance and the complexity of the embedded system functionality.

Nevertheless, if fast external response times of about 220µs are to guaranteed, the software overhead due to task-switching becomes a limiting performance factor. Moving functionality from software to hardware must be performed to solve the arising bottleneck. Therefore, exploitation of novel FPGA on-chip features like the SRAM capability leads to very efficient embedded system solutions. These issues will be addressed in this paper, which is organized as follows:

In chapter 2, we give a short overview of the state of the art and our own previous work. Chapter 3 introduces the emulation environment SPYDER-CORE-P1, which was developed for the emulation of sophisticated embedded systems. Chapter 4 describes the benchmark application and the different software tasks, which are running under the control of the RTOS *VxWorks*. Chapter 5 analyzes the performance effect and outlines the bottleneck. Chapter 6 analyzes different FPGA architectures and evaluates their special on-chip features. Chapter 7 the outlines the results of the investigation and chapter 8 summarizes the entire paper.

2 Previous Work

2.1 State of the Art

Hard real time requirements heavily influence the hardware/software partitioning in order to find a good solution with respect to performance and cost. The traditional method applied by most system designers today can be described as a two step solution. First, a printed circuit board containing all the selected chips is developed and after production, the necessary application software is written in the second step. The main disadvantage of this method is the lack of detailed knowledge about the

[1] This work was supported in part with funds from the Deutsche Forschungsgemeinschaft under reference number 3221040 within the priority program "Design and Design Methodology of Embedded Systems".

internal embedded system behavior, which can only be determined very late in the design process. If system requirements are not met, corrections must be made late in the development process significantly increasing design time and total costs. Sometimes, the entire project must be cancelled due to time or budget restrictions.

The more scientific approach tries to describe the behavior of a system at a high level of abstraction. After the design entry step, some performance evaluation is done followed by co-synthesis techniques for software and hardware as described in [6][7]. The result is verified using one or more combined simulators. This approach suffers from the lack of an appropriate design language to describe all the aspects of an embedded system. A mixture of different languages must therefore be used. Each language addresses a separate part of the system, which makes design entry a time consuming and difficult task. Additionally, the complex interaction between the system and its environment must be modelled for simulation, which is also very difficult and in some cases impossible without an unreasonable amount of resources, tools and design time.

If we compare the two design methods mentioned, we can say that using the traditional method leads to a system without detailed knowledge of behavior during the design process and is mainly based on the experience of the designers to make it work. The scientific approach recognizes that fact, but suffers itself from a lack of appropriate tools to apply the method. This was our motivation to search for an applicable method for getting detailed information based on understandable measurements of the system during an early design stage. Furthermore, the method should be very close to the final target system and avoid estimations of the embedded system behavior to decrease the risk involved with the development.

An approach, which is able to provide these features, is embedded system emulation. It offers the possibility to find the best hw/sw partitioning and the best usage of system resources early in the design process. Many different emulation environments exist. A few environments [8][9] are principally very flexible, but because of the high degree of flexibility are not suitable for an embedded system like the type of fast reactive embedded systems under consideration here. This is due to the great expansion of the resulting system, leading to the loss of real time capability. Therefore, we have developed the new emulation environment explained in chapter 3.

In the last section, we have clearly motivated the use of the emulation method, which allows to move functionality from software to hardware and visa versa, in order to find the best hw/sw partitioning. That capability is mainly based on a closely coupled microcontroller to one or more different FPGAs. One key feature when moving functionality from software to hardware is the appropriate memory needed for the data structures assigned to the function. Therefore, novel FPGA architectures provide different on-chip memory capabilities, which can be used to increase system integration and clock speed while decreasing I/O pin requirements, in contrast to providing off-chip memory. Principally, two different types of on-chip memory have emerged, which can be described with fine-grained and coarse-grained.

Fine-grained memory is available in FPGAs like the Xilinx XC4000 architecture [12]. Each lookup-table can be programmed as a small SRAM. These SRAMs can be combined for larger SRAM-arrays. Due to the fact, that SRAM can be implemented in each CLB, the SRAM-capability is distributed over the entire chip area, but consumes much CLB and routing resources if large SRAM memory is to be implemented. Another memory feature is provided by the Xilinx XC6000 architecture [13], in which each memory bit in a flip-flop is stored. The interesting feature, in contrast to the other architectures, is the fact that each flip-flop is embedded in the XC6000 chip architecture and can be accessed for read or write by a microcontroller, without any additional FPGA routing resources. That feature simplifies the exchange of intermediate results between a software task running on a microcontroller and a hardware task, which operates at the FPGA.

The coarse-grained approach is used in the Actel A3200DX FPGAs [15], the Altera 10K devices [14] and the Lattice ispLSI 6192 FPGAs [16]. Its basic characteristic is distinguished through dedicated SRAM blocks with array sizes from 2k up to 32kBit, which are embedded onto the FPGA architecture. Due to the fact that such large SRAM arrays are dedicated on-chip features, it can be used without any reduction of the programmable logic resources. If an application requires only small or no SRAM arrays, that chip area occupied by the SRAM is wasted. Current research work [17][18] tries to use such embedded SRAM as a large multi output ROM to implement logic, should the application do not require memory. This ongoing work is an interesting research topic, but is still not available today in standard VHDL-based development tools for FPGAs.

2.2 Our Basic Work

The past two years were marked by the development of innovative embedded systems in the area of industrial automation and communication. This was done in cooperation with several companies in which these embedded systems were used for industrial applications. This work serves as a basis for the current research work, which investigates more effective methods for embedded system design exploiting FPGAs.

The design of an Asynchronous Transfer Mode (ATM) diagnostic monitor was characterized by extremely fast external events of about 3µs, which must be recognized and counted. Dedicated hardware had to be developed, which consisted of many 32 bit wide counters. The counter values had to be read by a microcontroller. The comparison with different FPGA architectures for that application showed that the XC6000 architecture improves the hardware functionality for gates by about 60% in contrast to the XC4000 architecture. This improvement is mainly due to the XC6000 open architecture, which enables access to all application flip-flops without any additional routing resources. Therefore, the XC6000 FPGA is interesting for applications, which must exchange intermediate values between software and hardware, such as the application presented herein, further analyzed in chapter 6. The work about the ATM-diagnostic monitor [3] was done using our first generation emulation environment. A further contribution to the current work documented in this paper lies in the experience that emulation is a powerful method to analyze the internal behavior of complex embedded systems. It guided the definition of the architecture of our second generation emulation environment called SPYDER-CORE-P1 [4], which will be described in chapter 3. It was used for the presented work.

Another major project was done in cooperation with different industrial companies and led to the development of an Actuator Sensor Interface (ASI), a so-called ASI-Master. ASI is a new system which allows for the connection of up to 128 binary actuator and sensor devices with an appropriate control unit via a single bifilar cabel. An additional key feature of this

work is the global access to the ASI-Master via the Internet, which leads to value-added services as described in [5]. Currently that project uses the RTOS *VxWorks*, which is running on SPYDER-CORE-P1. It is used for the scheduling of different tasks. This application serves as a benchmark and is explained in chapter 4. That application example was selected due to three major characteristics:

1. Sophisticated software task architecture controlled by the RTOS.
2. Novel microcontroller architecture in combination with different FPGA architectures.
3. Fast reaction times to external events.
4. Sensitive hw/sw partitioning, which heavily influences the entire system performance.

These characteristics make up a complex internal system behavior, therefore being suitable to demonstrate the benefits of the emulation method and the benefit gained by exploitation of FPGA on-chip features. The presented work gives answers to three important questions which arise from a system designer´s point of view:

1. What is the minimum clock speed in order to meet all real time constraints?
2. How much computation performance is consumed by the RTOS in a fast reactive system in that case?
3. What is the best hw/sw partitioning to solve the arising bottleneck mainly caused by the RTOS?
4. What is therefore the best FPGA architecture to implement the hardware part, especially for the exploitation of its on-chip features?

This paper shows that these questions are not limited to that specific application, but are general questions about many applications in the mentioned area.

3 Emulation Platform SPYDER-CORE-P1

The current version of the SPYDER emulation environment consists of two boards. SPYDER-ASIC-X1 [10] is one part of the tool set and mainly addresses the emulation of large VHDL-based ASICs designs on a FPGA. SPYDER-CORE-P1 is the second part of the tool set and is used for the emulation of the entire embedded system architecture. Both boards are compatible and can easily be connected to each other via a backplane. For the work described here, only SPYDER-CORE-P1 was used, thus only this board is described in detail (see figure 1). 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.

A 32 bit RISC embedded PowerPC 403GA/GCX microcontroller [2] is used. It provides the following two advantages:

1. It is available in a wide clock frequency range. This enables the emulation of a large area of applications. An analysis step can achieve the right values to satisfy all real time requirements and prevent an oversized or undersized system.
2. A flexible programmable bus interface provides a direct interface to most peripheral devices. This enables the simple implementation of a low or high end system without additional glue logic.

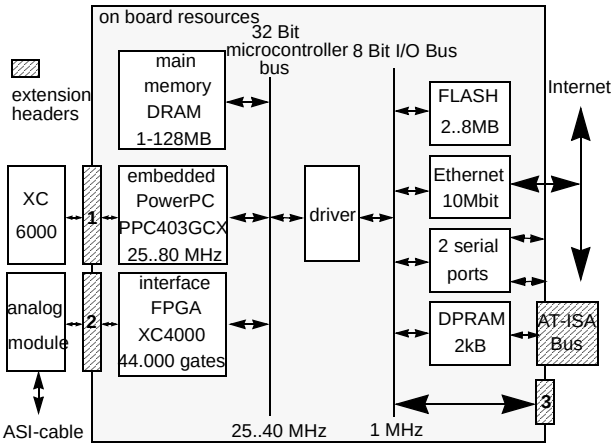

Figure 1. Architecture of SPYDER-CORE-P1

The high-speed microcontroller bus is connected to the main memory and the interface FPGA XC4000. 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 hardware. Each microcontroller bus signal is available on the extension header 1. They provide the means to connect a co-processor, e.g. a reconfigurable XC6000 FPGA, to the microcontroller very closely and to connect debugging equipment, e.g. logic analyzer, to the SPYDER-CORE-P1 emulation environment.

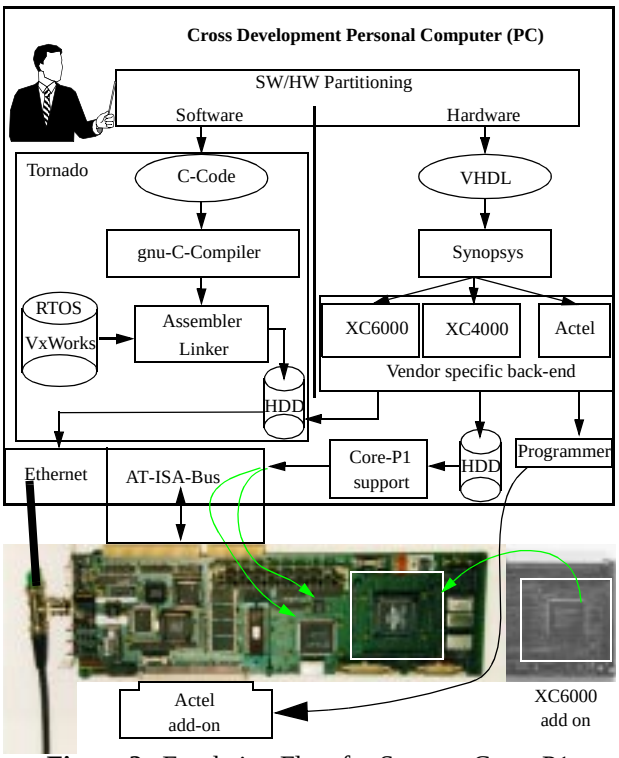

Figure 2. Emulation Flow for SPYDER-CORE-P1

Figure 2 shows the emulation flow for the SPYDER-CORE-P1 environment. The sw/hw partitioning is done by the designer. The emulation starts with an initial partitioning and can rapidly be changed based on the detailed view account obtained

during the emulation. The software path is supported by the RTOS design environment, called *Tornado*. Different Tasks written in C-code must be compiled using the *gnu-C*-Compiler. The *Tornado* environment links the tasks together with the RTOS routines and store the code on the hard disk. A dedicated loader transfers the code to the target board via a 10MBit Ethernet (TCP/IP) link.

The hardware parts can be described in VHDL and synthesized into a netlist representation using the *Synopsys Design Compiler*. The place and route step is performed by the vendor specific back-end tools. The XC6000 design data can be remotely accessed by the microcontroller via Ethernet from the PC hard disk, using appropriate *VxWorks* remote access features. The configuration is done by the microcontroller using the XC6000 32 Bit fast microcontroller interface. The configuration data of the XC4000 devices can be downloaded to the SPYDER-CORE-P1 board to both a serial EEROM and the XC4000 FPGA itself using CORE-P1 support software tools, which transfers the bit stream via the AT-ISA-Bus. Actel devices must be programmed using Actel's programming equipment and can be added to the SPYDER-CORE-P1 emulation environment using the extension header 2 and a dedicated device specific add-on board.

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 microcontoller.

The same architecture shown in figure 1 was used to emulate the two applications mentioned in chapter 2.2. The SPYDER-CORE-P1 environment was modified by only connecting a small add-on board with application specific hardware (ASIC or sub-module) to the extension header 2. In the case of the ATM-project, a so-called Adpation-Layer-Controller (ALC) was added and for the ASI-project, a dedicated analog module was used.

4 Internet controlled ASI-Master

The Actuator-Sensor-Interface (ASI) connects up to 32 ASI slave chips via a single, bifilar cable with a ASI master unit (see figure 3).

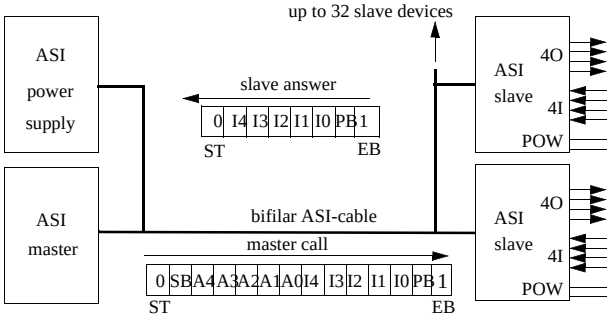

Figure 3. ASI Communication System

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 the master call. The ASI slaves respond if the address in the master call matches their own address, and transfers the input data image back to the master. Each slave is able to connect up to four binary sensors (4I) and up to four (4O) actuators. The serial master call protocol must be modulated on the ASI cable using a dedicated analog module (see figure 1). An ASI slave is available as a single chip solution which has the same capability on-chip. Additionally, an ASI power supply unit provides 24 DC voltage for the slaves via the same cable.

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 using different FPGA architectures (see figure 4). The analysis results are shown in chapter 6. 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.

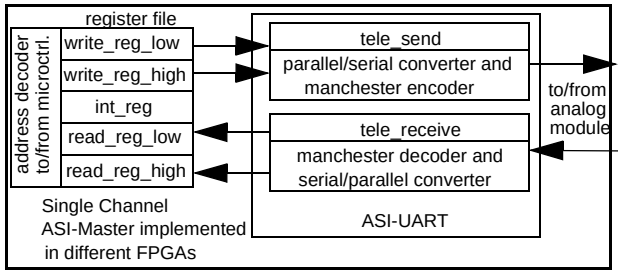

Figure 4. Initial ASI-Interface Hardware

The RTOS *VxWorks* [11] runs on the SPYDER-CORE-P1 emulation platform. It provides a TCP/IP stack for communication via ethernet. Additionally, this interface is used as a link to the *VxWorks* development environment *Tornado* running on a host PC.

During the initial hw/sw partitioning, four tasks run on the RTOS; two are hard real-time tasks and two tasks have no real-time constraints (see figure 5).

1. 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.
2. The *Control* task is also a hard real-time constraint task and uses the current process data image to calculate the control commands.
3. The server task has no real-time requirements and is responsible for data and command exchange via the Internet.
4. The embedded *C_Server* task also has no real-time requirements and transfers commands between a JAVA applet running on the calling client computer and the ASI-Master.

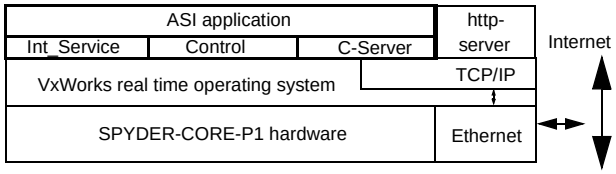

Figure 5. Initial Software Architecture

For a more detailed description, refer to [5].

5 Embedded System Performance Analysis

Figure 6 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 interface hardware implemented (see figure 4) in a FPGA and the microcontroller. The *Int-Service* and *Control* tasks mark their beginning and end states by accessing a simple I/O device (I/O-signal), recorded by the logic analyzer. The ASI standard defines the time delay between two 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 must be made. Figure 6 shows that 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.

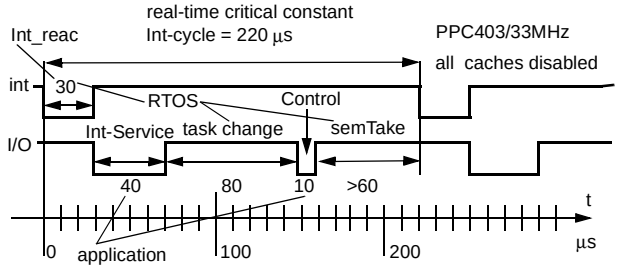

Figure 6. Execution Times

Figure 7 shows the clock frequency range from 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 (see figure 5) and the real-time critical constant. The curve depicts the case without 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, it is still able to guarantee the real-time requirement. If the clock frequency decreases, the y-value increases above one. This means that the microcontroller performance is undersized.

The ASI-Communication System in figure 3 and the ASI-Interface Hardware in figure 4 shows for simplicity the system architecture for one ASI-Channel. That means, one ASI-cable with up to 32 slaves is connected to the ASI-Master and therefore it is called a Single-ASI-Master. Figure 7 shows, that for this case the microcontroller can met the real-time requirement with a minimum clock speed of 33 MHz. Current ASI-development projects focus on connecting two, three or four ASI-cables to a common ASI-Master unit in order to extend the number of slaves by the appropriate factor of two, three or four with only a slightly increasing of the total system costs.

If two ASI-cables are connected to the ASI-Master (called Dual-ASI-Master), the same real-time critical task sequence (see figure 6) must be executed in half of the time given by the real-time critical constant (220µs) in order to meet all real-time requirements for both ASI-channels. In this case, as indicated by the dashed line at the y-value of 0.5 in figure 7, the real-time requirements can not be satisfied although the highest available clock speed of 80Mhz is used. This short explanation motivates clearly the effort, which were spent to determine the minimum clock speed for a Single-ASI-Master and the resulting system behavior, if additional ASI-channels would be added.

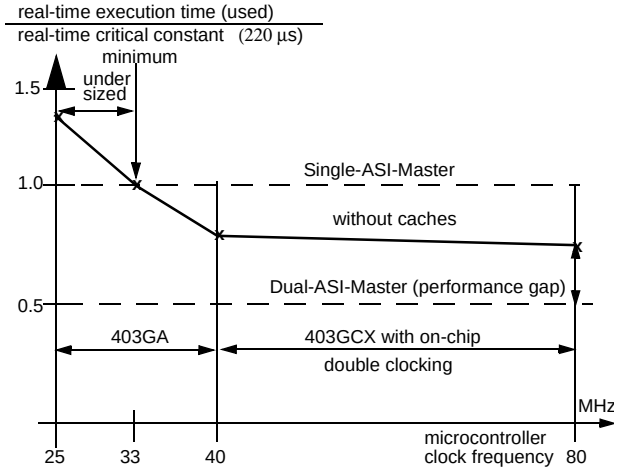

Figure 7. System Execution Resources

At this point the embedded system emulation shows that a state of the art RTOS achieves task switching times of about 80µs and interrupt reaction times of 27µs. As shown in figure 6, 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) plus the interrupt-reaction time (here 30µs), the software overhead of the RTOS becomes a limiting factor for the total embedded system performance.
- Only a Single-ASI-Master can be realized within the available microcontroller clock range. A Dual-ASI-Master (or more) can not be implemented with a system solution executing a *Int_Service* routine in software.
- That fact forces the *Int_Service* routine running at the microcontroller to be moved from software to hardware. That functionality must be implemented on a FPGA in addition to the described ASI-Interface hardware depicted in figure 4, and is described in the next sub-chapter.

5.1 HW/SW Partitioning to solve the bottleneck

Figure 8 shows on the left side the flow diagram of the *Int_Service* task. If the microcontroller receives an interrupt (every 220µs), it updates the current input slave data from the process, and sends the next output slave data to the process. If all connected ASI-slaves in the polling cycle has been processed, the entire process data image is generated. If the maximum of 32 slaves are used, the process data image can be generated in about 7ms. The *Int_Service* task, which runs in software, holds the process data image in the main memory of the microcontroller. If that functionality should be moved to hardware, a

memory for the input data image with total size of 192 bits (6bit x 32) and a memory for the output data image with a total size of 224 bits (7bit x32) must be provided on the target FPGA chip. Additionally, the sequential control flow of the software task must be implemented in hardware using a Finite State Machine (FSM).

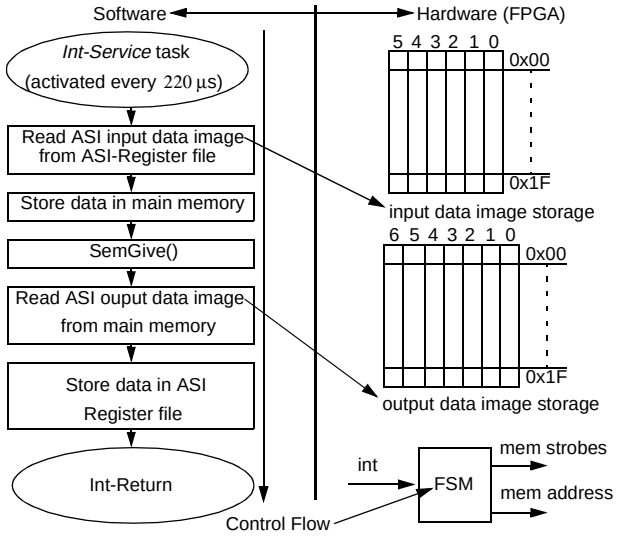

Figure 8. *Int-Service* shift from Software to Hardware

As depicted in figure 9, two on-chip memories with single port read and single port write capability, and a FSM must be implemented to realize the entire *Int_Service* routine in hardware.

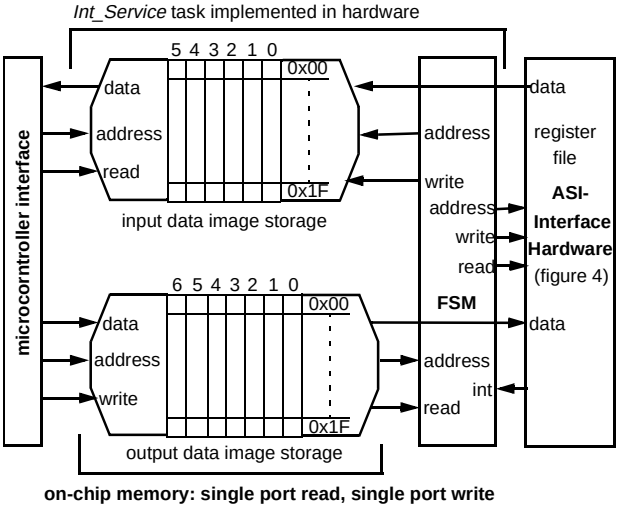

Figure 9. Architecture of the new FPGA hardware

That additional hardware part is located between the initial ASI-Interface hardware shown in detail in figure 4 and a standard 8 bit microcontroller interface on the other side. The FSM is responsible in transfering the receive data from the slave answer into the input data image memory and to copy the current output data image to the ASI-Interface hardware for the next master call.

The new FPGA hardware architecture is able to free the microcontroller from responding to the fast external events every 220µs. Most applications require the evaluation of the entire process image every 10ms in order to statisfy real-time constraints. That means, if the *Control* task read out the input data image every 10ms, the task-switching decreases by a factor of 45 (10ms/220µs) in contrast to the case with the software *Int_Service* routine. That leads to the fact that the RTOS must not perform so many task switches, which results in a dramatic decrease of execution performance consumption by the RTOS. That resources can be used to realize a Dual-ASI-Master and global network communication (see figure 10).

6 Analysis of different FPGA architectures

The emulation of the ASI-Master shows that the efficiency of the hardware implementation depends greatly on the exploitation of on-chip features provided by different FPGA architectures. For simplicity, the FPGA-resources for a Single-ASI-Master were depicted. For more than one ASI-channels, the FPGA-resources increases linear. The used hardware architecture (depicted in figure 9) serves as a benchmark and were therefore be divided in three major parts:

1. The initial ASI-Interface hardware, which consists mainly of finite state machines in combination with many counters.
2. The FSM for the control flow implementation of the interrupt handling.
3. The two memory arrays for the input (192 bits) and output data (224 bits) image, which must provide simultaneously single port read and single port write capability.

The SPYDER-CORE-P1 emulation environment offers three different FPGA (see figure 2) architectures for the analysis. The following results were obtained:

1. If the XC4000 architecture was used, the initial ASI-Interface hardware part needs 166 CLBs and the *Int_Service*-FSM needs 20 CLBs. The lookup tables were used as fine-grained on-chip SRAM. If the simultaneous single port read and single port write mode is selected, 16 bits were implemented in one CLB. This means, a total of 26 CLB must be use for both memory arrays. The ratio between the additionally implemented *Int_Service* part and the initial ASI-Interface hardware (46 CLB / 166 CLB) is about 28%.
2. If the XC6000 architecture was used to implement the initial ASI-Interface hardware part, 841 XC600-cells were used. The *Int_Service*-FSM needs 112 cells. The memory array must be implemented with flip-flops, which are embedded in the XC6000 architecture. This means, no routing resources for the access via the microcontroller are necessary. The simultaneous single port read and single port write capability must be realized with additional logic cell and occupies a total amount of 957 cells. The ratio between the additional implemented *Int_Service* part and the initial ASI-Interface hardware (1069 cells / 841 cells) is about 127%.
3. If the Actel A1200XL architecture was used to implement the initial ASI-Interface hardware part, 436 Actel logic modules were used. The *Int_Service*-FSM needs 51 logic modules. The Actel FPGAs provide coarse-grained embedded on-chip SRAM in devices greather than A32100DX with a minumum of 1362 logic modules. For applications which require much less than the A32100DX capacity, only flip-flops are available to

realize on-chip memory, such as this application, which leads to 763 logic modules for the appropriate memory arrays. The ratio between the additional implemented *Int_Service* part and the initial ASI-Interface hardware (814 logic modules / 436 logic modules) is about 187%.

All values were obtained using the same VHDL design description and the standard development tools as depicted in figure 2.

7 Results

The result of the ASI-Master emulation using the SPYDER-CORE-P1 environment gives answers to the questions mentioned at the end of chapter 2.1, which arises from to system designer's point of view:

1. The minimum clock frequency is determined by 33MHz in order to guarantee all real-time constraints under worst-case conditions for a Single-ASI-Master.
2. At 33MHz, the RTOS consumes 77% of the total execution performance, which is very inefficient due to the fact, that the external response time of 220µs operates at the same order of magnitude as the RTOS task-switching time.
3. In order to solve that bottleneck, a change of the initial hw/sw partitioning must be done. The *Int_Service* routine must be moved from software to hardware.
4. The investigation of different FPGA architectures shows that the efficiency of the implementation is greatly influenced by the exploitation of the on-chip memory capability as summarized in table 1.

	XC4000 (CLBs)	XC6000 (cells)	Actel (logic modules)
initial ASI-Interface hw	166	841	436
Int_Service FSM	20	112	51
Int_Service on-chip memory	26	957	763
additional resources for *Int_Service* (FSM, memory)	28%	127%	187%

Table 1. Architecture Implementation Results

The column for the XC4000 architecture shows clearly, that the exploitation of the fine-grained on-chip SRAM capability led to a very efficient implementation of the *Int_Service* functionality. The overall amount of the additional hardware parts is only increased by 28%, mainly due to the efficient implementation of the memory. This architecture is best suited for applications in the lower or medium gate range up to 10.000 gates, which needs on-chip SRAM capability of a few hundred of bits. This type of application is not limited to the specific benchmark presented in this paper, but is very common in a wide range of embedded system applications.

The column for the XC6000 architecture shows that the increase for the additional *Int_Service* hardware rises up to 127%. This is due to the fact that no SRAM capability is available and the memory must be implemented with flip-flops like the Actel A1200XL architecture. In contrast to the Actel chips, the XC6000 benefits from the fact that all flip-flops are embedded in the chip architecture and can be accessed by the microcontroller, without any additional routing and logic resources. In applications with many counters or FSMs, which must exchange intermediate results with the microcontroller, that fact can become a major effect and contribute a significant benefit as described in [3].

The column for the Actel A1200XL architecture shows that the increase for the additional *Int_Service* hardware resources rises up to 187%. The main disadvantage is the lack of on-chip features like SRAM in the gate range up to 10.000 gates. That forces the designer to use flip-flops for the memory, which is very inefficient. The wide application area below 10.000 gates should be supported with on-chip SRAM features in the future. This would make that architecture very interesting, especially in combination with the advantage of antifuse based FPGAs, which do not need external memory for the configuration data.

7.1 Consequences for the Application

Moving the *Int_Service* Task to hardware leads to better exploitation of the hardware resources. It reduces the software overhead of the RTOS running on the microcontroller. The positive consequence is that most of the execution performance of the microcontroller can be used for the application. The final embedded system architecture is shown in figure 10.

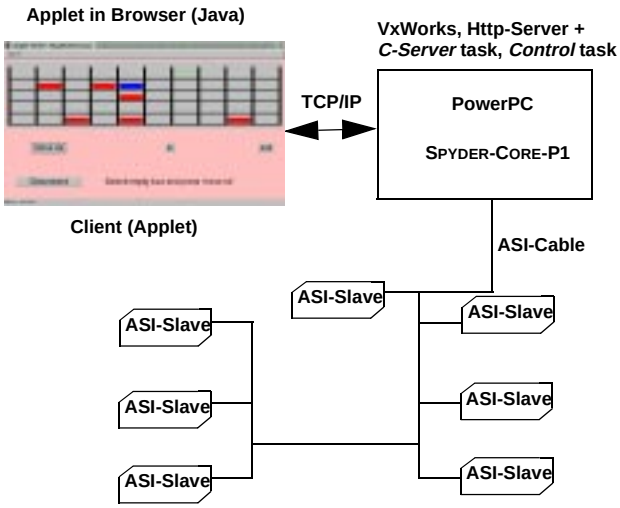

Figure 10. Embedded System Architecture

A Java applet is running on a client computer and connects via TCP/IP with the *C-Server* task running on the PowerPC microcontroller in the SPYDER-CORE-P1 environment. This link is used to exchange commands for the *Control* task, which controls an industrial plant. To do this, it uses the current process data image generated by the ASI-Master every 7ms.

To demonstrate the entire system, we use a model of an industrial shelf as shown in figure 11. It can be controlled by 28 sensors for the current x,y,z position of the shelf wagon and 11 actuators for the activation of the electrical motors to move the wagon.

For all sensors and actuators, a total amount of seven ASI-slaves (4I/4O) must be connected to the ASI-cable.

A operator can submit commands from his own PC, which are transfered to the industrial shelf via the internet. A camera captures the life image from the shelf and reports it back to the

operator. With this, the operator can observe the execution of his commands online. If you are interested in a demonstration of this application, please send a email to weiss@fzi.de or stecki@fzi.de. We will setup an appropriate user-id and password for your login. All you need for the evaluation is a running www-browser and internet access.

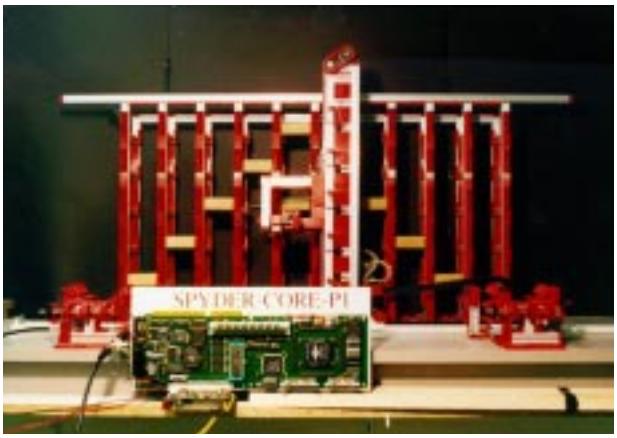

Figure 11. Demonstrator: Industrial Shelf Model

Note that this additional feature, which we call value-added services in the industrial automation [5], can therefore be implemented without increasing the microcontroller performance. It is mainly based on the determination of the best hw/sw partitioning found through emulation and the exploitation of the FPGA on-chip features.

8 Summary

System performance is influenced by many indeterministic parameters. In order to meet all real-time requirements and exploit all available system resources, a detailed knowledge of the internal behavior is necessary. Therefore, this paper presents the emulation of a fast reactive embedded system, running under control of a RTOS, as a powerful method to solve the problems mentioned and shorten the design time and risks.

In this paper, a novel emulation environment called SPYDER was introduced, emulating very close to the final target system. A industrial benchmark design was used to demonstrate this method. The minimum working frequency lies at 33MHz and the emulation shows that an RTOS consume 77% of the total execution resources, if the reaction times decreases down to the same delay as the task-switching times.

In order to solve that bottleneck, the initial hw/sw partitioning based on an *Int_Service* routine in software must be changed and moved to hardware. That additional hardware implementation needs a few hundred bits of on-chip memory in combination with an FSM. The analysis of different FPGA architectures shows that the efficiency of the implementation depends greatly on the exploitation of their on-chip features, such as SRAM. For the presented benchmark application, which is very common in a wide range of embedded systems, the XC4000 architecture with its fine-grained on-chip SRAM feature is best suited. It is exploited to implement the *Int_Service* routine in hardware. Therefore, the additionally used CLB resources are only 28%. The other architectures show much higher consumptions, of up to 187%, mainly based of the lack of appropriate on-chip memory features in the lower and medium gate range of up to 10.000 gates.

References:

[1] W. Wolf: *Hardware-Software Co-Design of Embedded Systems.* Proceedings of the IEEE, Vol. 82, No.7, July 1994.

[2] IBM: *PPC403GA/GCX Embedded Controller - User Manual.* IBM Corporation 1997

[3] Karlheinz Weiß, Ronny Kistner, W. Rosenstiel: *Analysis of the XC6000 Architecture for Embedded System Design.* Field-Progammable Custom Computing Machines (FCCM), Napa Valley CA, April 1998,

[4] Karlheinz Weiss, Thorsten Steckstor, Carsten Oetker, Ronny Kistner: *Data Sheet and User Manuel SPYDER-CORE-P1*. http://www.fzi.de/weiss.html, FZI & University Tübingen 1998

[5] A. Hergenhan, Christoph Weiler, Karlheinz Weiß, Wolfgang Rosenstiel: *Value-Added Services in the Industrial Automation.* ACoS'98, Lisabon Portugal, April 1998

[6] Sanjaya Kumar, James H. Aylor, Barry W. Johnson, Wm. A. Wulf: *The Codesign of Embedded Systems.* Kluwer Academic Publishers, 1996

[7] Rajesh Kumar Gupta: *Co-Synthesis of Hardware and Software for Digital Embedded Systems.* Kluwer Academic Publishers, 1995

[8] S. Hauck, G. Borriello, C. Ebeling: Springbok: *A Rapid-Prototyping System for Board-Level Designs.* ACM/SIGDA 2nd. International Workshop an Field-Programmable Gate Arrays, Berkley, Feb. 1994.

[9] G. Koch, U. Kebschull, W. Rosenstiel: *A Prototyping Environment for Hardware/Software Codesign in the COBRA Project.* Third International Workshop on Hardware/Software Codesign, 1994.

[10] Karlheinz Weiss, Thorsten Steckstor, Carsten Oetker, Ronny Kistner: *Data Sheet and User Manuel SPYDER-ASIC-X1*. http://www.fzi.de/weiss.html, FZI & University Tübingen 1998

[11] *VxWorks Reference Manual and Programmer's Guide*. WindRiver Systems, Edition 1, 1997

[12] Xilinx Inc.: *XC4000 (E, XL, XV) User Manual and Application Notes.* http://www.xilinx.com, 1998

[13] Xilinx Inc.: *XC6000 Field Programmable Gate Arrays.* http://www.xilinx.com, 1996.

[14] Altera Corporation: *Altera 10k Databook*, http://www.altera.com, 1996

[15] Actel Corporation: 3*200DX Field Programmable Gate Arrays*, http://www.actel.com, 1995

[16] Lattice Corporation: *Data Sheet: ispLSI and pLSI 6192 High Density Programmable Logic with Dedicated Memory and Register/Counter Modules*, http://www.lattice.com, 1996

[17] Steven J. E. Wilton: SMAP: *Heterogenous Technology Mapping for Area Reduction in FPGAs with Embedded Memory Arrays*, ACM/SIGMA International Symposium on FPGA 98

[18] Jason Cong, Songjie Xu: *Technology Mapping for FPGAs with Embedded Memory Blocks*, ACM/SIGMA International Symposium on FPGA 98