A Configurable Target Architecture for Rapid Prototyping High Performance Control Systems*

Franz Fischer Thomas Kolloch Annette Muth Georg Färber
Laboratory for Process Control and Real–Time Systems
Prof. Dr.–Ing. Georg Färber
Technische Universität München, D–80290 München, Germany
Phone: +49–89–289–23550, Fax: +49–89–289–23555
{Franz.Fischer,Thomas.Kolloch,Annette.Muth,Georg.Faerber}@lpr.e-technik.tu-muenchen.de

Abstract *Embedded hard real–time control systems show growing functional complexity as well as increasing demand for short response times and high computing performance. This paper presents a target system architecture suitable for rapid prototyping embedded real–time applications. The tightly coupled heterogeneous multiprocessor system includes state–of–the–art high performance microprocessors as well as Configurable I/O processors based on field programmable gate arrays. It is designed to allow better performance guarantees by simplifying the prediction of tight worst case execution times, which are the base for the schedulability analysis of the tasks executed on the microprocessor based nodes. The Configurable I/O processors are used for both interacting with the embedding system and processing tasks with deadlines too short to be met by software. The automated transformation of the system specification and the possibility to connect the prototype to the embedding process allows the validation of the specification regarding function as well as timing constraints.*

Keywords: embedded control system, rapid prototyping, HW/SW–Codesign, hard real–time, schedulability analysis

*This work was supported with funds of the *Deutsche Forschungsgemeinschaft* under reference number Fa 109/11-1 within the priority program "Rapid Prototyping for Embedded Hard Real–Time Systems."

1 Introduction

It is generally agreed that the rapid prototyping methodology reduces the development effort and therefore the "time to market" for a new embedded controller application. In the domain of hard real–time systems, as used e. g. in automotive control and mechatronics, an adapted target platform should support both the proof that the system meets all its deadlines, and the automated translation of a system specification into an executable prototype. This prototype architecture replaces the small scale embedded parallel system during the specification phase.

1.1 Design Process

Using a HW/SW–Codesign methodology (Figure 1), the rapid prototyping design process is divided into the following steps: First, we start to specify the embedded system using implementation independent languages like SDL to express system structure, behavior and dynamic. Concurrently we describe the embedding process with event–streams, timing constraints (deadlines) and event dependence matrices (Section 1.3).

After the classification according to computation complexity and required response times (Section 1.2) an automatic translation from SDL to a (C–)task system (SW–part) or a SDL to VHDL conversion (HW–part) occurs. Be-

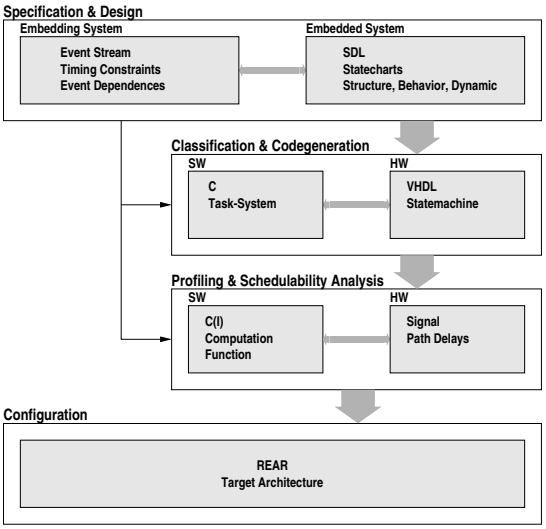

Figure 1: Rapid Prototyping Design Process

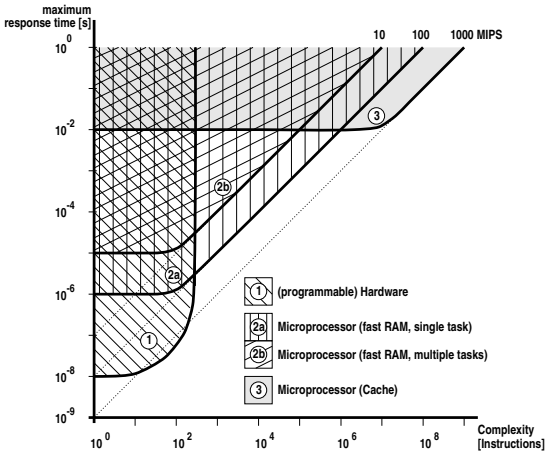

Figure 2: Classification of application tasks

cause of the difficulties of mapping an asynchronous SDL–specification to a synchronous VHDL–description, we — in the moment — have to redesign the HW–part with Statecharts and then translate this system model to VHDL. For synthesizing the HW part of the system model, the design cycle currently uses the commercial tools Statemate (i-Logix) for system specification and VHDL–code generation, Synopsys for synthesis and Xilinx XAct for fitting the netlists into the target technology.

The next step includes the compilation and synthesis of the C– and VHDL–codes. Subsequently the WCETs and signal delays are derived. They are the source information for the schedulability analysis described in section 1.3.

In a final configuration step, the whole system model is mapped to the prototyping target architecture.

1.2 Target Architecture

Especially with modern high performance microprocessors and multi level memory hierarchies the estimation of WCETs is a non–trivial problem and the numbers determined are often too pessimistic and hence unusable. Life can be made a little easier, if "appropriate" processing nodes are used for the different application tasks with their specific requirements regarding maximum response time $t_{R,max}$ (the tasks deadline), maximum amount of computation C (complexity) and the code and data size of the task. Figure 2 shows the estimated achievable deadlines on different types of processing units for a given amount of computation.

For a task implemented in programmable hardware response times are usually short (in the order of some 10 ns) and can be precisely specified, while the complexity of the functions to be performed is limited (Curve 1). Tasks implemented in software can be as complex as necessary but response times are longer and at the same time harder to predict. With a microprocessor system specially tuned for real–time applications — a Real–Time Unit (RTU), — response times down to 1 μs can be achieved and determined with reasonable accuracy (Curves 2a and 2b). A modern standard computer system — a High Performance Unit (HPU) — on the other hand is more powerful but tight deadlines are much harder to guarantee (Curve 3).

Our target system architecture REAR (Rapid Prototyping Environment for Advanced Real–Time Systems) consists of these different types of processing nodes outlined above to achieve the flexibility, configurability and predictability of usable WCETs necessary

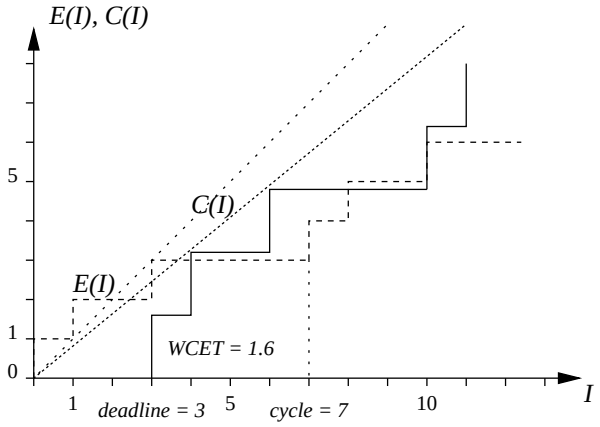

Figure 3: Example of Event Function $E(I)$ and resulting $C(I)$

for rapid prototyping hard real–time applications.

1.3 Schedulability Analysis

Proving that a system meets all its deadlines even in the worst case — i. e. guaranteeing its performance — is a key requirement of hard real–time systems. As opposed to "soft real–time", "hard real–time" means that a deadline miss may result in loss of lives and money. We do not want to test the timely response by "trial and error", but rather verify the necessary properties of the reactive system by the schedulability analysis developed by Gresser [1, 2, 3]. In contrast to analysis methods for time driven (periodic) systems where task deadlines are guaranteed by the construction of the time driven schedule, this method takes into account the timely behaviour of the technical process which stimulates the event driven system's tasks.

Event streams describe the maximum possible number of events of a certain type within an interval I and lead to an *Event Function* $E(I)$. Single tasks are characterized by their worst case execution times (WCET) and the respective deadlines (maximum allowed response times) for the triggering events. The $C(I)$ Function is defined as maximum computation time requested and due within interval I. For a single task $C_i(I)$ can be calculated easily from $E(I)$ by shifting by the deadline and multiplication with the WCET (Figure 3).

While the resulting $C(I)$ for a number of *independent* tasks on a computing node is simply the sum of all the $C_i(I)$ functions, Gresser developed an algorithm to determine $C(I)$ for a network of communicating tasks, taking into account dependences of the triggering events, precedence constraints, inter node communication and mutual exclusion. For earliest deadline first scheduling he proved, that all tasks on one node meet their deadlines if the resulting $C(I)$ always runs under the bisector which specifies the available computing time in each interval.

The task model used by Gresser is similar to that presented in [4], but Jeffay does not consider event dependences and limits the analysis to single processor systems.

The paper is organized as follows: The next section surveys related work, in section 3 we present a detailed description of the REAR hardware architecture, especially focusing on the *Real–Time Unit* (RTU) and the *Configurable I/O Processor* (CIOP). Two example applications, a CAN controller and a stepper motor controller for a small robot arm are outlined in Section 4. In the last Section we summarize our results and indicate future experiments.

2 Related Work

As this paper focuses on target architectures, this section compares the basic approaches of related HW/SW–Codesign projects. The common idea is that the target architecture has to support the design automation of the embedded controllers with their appropriate software.

Optimizing the communication structure according to the bandwidth requirements, the "multi–layered architecture template" of SIERA [5] supplies different layers for the various protocols. Highest throughput can be found in the lowest layers using dedicated hardware (ASICs and FPGAs), while software is

dominant in the lower–rate upper layers.

The use of a small microcontroller with a FPGA–coprocessor is the basis of the POLIS [6] design automation system. The main research goals of this project are to preserve the semantics of their Codesign Finite State Machine (CFSM) specification during system synthesis and to improve the efficiency of the generated code.

The two design systems COSYMA [7] and VULCAN [8] try to minimize the unit costs by implementing as much as possible of the system specification in software, using a microcontroller as processing unit. Both projects move time critical system parts to hardware, i. e. programmable logic. Starting with the whole system in software, COSYMA uses application specific HW — the so called "coprocessor" — to increase the system performance and to meet the deadlines. VULCAN starts with a HW–solution and then extracts system parts to SW, as long as the timing constraints are satisfied.

Implementing the complete system in application specific HW is the concept of the CASTLE [9] design environment. The system is first realized using a Multi–FPGA board for test and HW/SW co–simulation. After the function and timeliness proof, the common specification can be compiled into an ASIC.

To optimize the performance/price relation, the Chinook system [10] supports the automated synthesis of I/O interfaces and the glue logic necessary to connect them to the general purpose microcontrollers of their target system. Device driver generation for the SW part is provided as well.

Supporting the verification of the functionality and timeliness of the real–time system, rather than minimizing the unit costs is the main objective of the REAR architecture. The automated transformation of the system specification and the possibility to connect the prototype to the embedding process should answer the questions "Is the specification of our application correct?" and "Is our analysis of the timing constraints correct?". The implementation of the now verified system specification with reduced production costs using architecture templates is the focus of our recently started research project ATLAS.

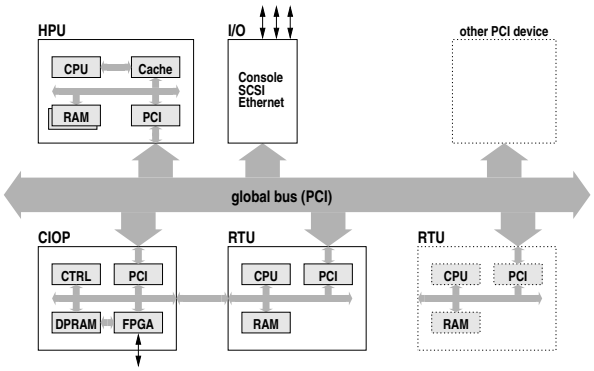

Figure 4: REAR hardware architecture

3 REAR Hardware Architecture

The target system is a configurable and scalable heterogeneous multiprocessor system. It consists of processing nodes with state–of–the–art high performance microprocessors (CISC and RISC type) and Configurable I/O Processors (CIOP) based on field programmable gate arrays. The nodes are tightly coupled by a global bus, which offers high throughput and low latency, both necessary to yield good results from the schedulability analysis. Figure 4 gives an overview of the target system architecture, which is mostly built from off–the–shelf components and uses PCI bus to connect the processing nodes.

If the need arises, other types of PCI cards like peripheral devices or DSP boards can be easily added to our three standard types of processing nodes described in the following subsections.

3.1 High Performance Unit

The High Performance Unit (HPU) is intended to run application tasks with a very high demand for computing performance as well as for memory space (e. g. image processing tasks).

Therefore an off–the–shelf computer system is used as HPU, equipped with a latest generation processor, large memory and typically two or three levels of caches for high *average* performance.

Due to the multi level memory hierarchy, superscalar processor architectures and speculative execution (as e. g. on the PentiumPro) the worst case timing analysis necessary for the schedulability analysis for hard real–time tasks is very difficult or results in unrealistic and unusable values. This problem can be alleviated partly by cache partitioning and locking techniques or by avoiding frequent interrupts and context switches [11]. The effect of these techniques will be investigated using multiple computing intensive tasks and sporadic interrupts as a workload for a real–time operating system on the HPU.

Currently we use a 133 MHz Intel Pentium based system running Linux for software development and soft real–time tasks.

3.2 Real–Time Unit

The Real–Time Unit (RTU) is designed to run application tasks with medium complexity, limited memory requirements but short deadlines. In order to simplify the prediction of WCETs, this processing unit is ideally based on a processor with an easy to analyze pipeline and uses fast static main memory and no or only a 1st level cache.

For the RTU we chose the MIPS Orion processor family (R4600/R4700), a successor of the MIPS R3000. Both architectures are often used in high performance embedded systems like network components, printers or automotive control systems. Yerang Hur et. al. [12] describe a methodology for worst case timing analysis for the R3000 processor which yields reasonably good predictions. Their *extended timing schema* provides means to reason about the execution time variation of a program construct influenced by the surrounding program constructs. To calculate the number of cycles used by the instructions within a basic block, they build a reservation table of all utilized pipeline stages. An algebra with concatenation and prune operations allows the recursive analysis of the whole program. To determine the caching effects of the memory references in the execution path, information about the cache contents left by preceding memory accesses is stored too. One flaw of this methodology is that effects caused by interrupts and task switches are not considered. By disabling task preemption — this would lengthen the response times — we can solve this problem in a first step.

Table 1: Real–Time Unit memory throughput

Throughput / MBs^{-1}	single access		burst access	
	read	write	read	write
L1 I+D caches	1200	800	1200	800
SRAM module	100	133	229	267
DRAM	50	57	114	123

The RTU was built using the Galileo 4 evaluation system, which integrates the Orion CPU, memory, serial ports and the PCI interface on a single PCI card. To narrow the memory bandwidth gap between the CPU and the DRAM available on the board a SRAM module was built from synchronous cache RAMs and a Xilinx 7336 CPLD which implements the R4600 system interface [13]. This module can be plugged between the CPU and the Galileo board and serves read and write requests from the CPU without wait states. Table 1 compares the peak throughput for DRAM, the SRAM module and the processors L1 caches for a R4600 with 100 MHz pipeline and 50 MHz system interface clock. Performance measurements for application and real–time operating system code will be done in the near future.

One additional CPLD implements the glue logic to connect the CIOP to the Real–Time Unit's local bus and eight hardware spinlocks. These simplify the implementation of low overhead multiprocessor communication primitives within the operating systems running on the microprocessor based computing nodes. Depending on the access address, one of eight

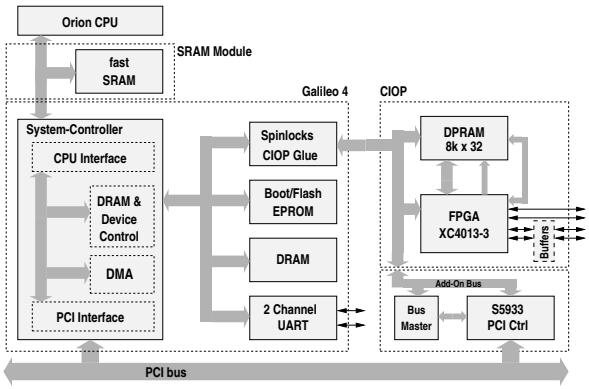

Figure 5: RTU with Configurable I/O Processor

spinlocks can be either tested or tested and set in a single read access, whereas a write clears the spinlock. Figure 5 depicts a block diagram of RTU and CIOP.

3.3 Configurable I/O Processor

The CIOP serves two purposes: Linking the prototyping architecture to the embedding process and performing tasks with deadlines too short to be met by software. Within our application domain we expect this to be event triggered I/O tasks and not computation intensive tasks. Examples include, but are not limited to high speed serial signals, high data rate parallel connections and time functions like pulse width modulation. Hence the CIOP is not a *coprocessor* — compared with the processing power of the RTU and HPU processors, the communication latency would eat up the performance gain for most applications — but a *preprocessor* which acts as an "event filter" and may reduce the load on the busses and the number of interrupts and context switches on the RTU and HPU.

The Configurable I/O Processor consists of one Xilinx 4013E Logic Cell Array (replaceable by a pin compatible XC4025E with more chip resources), additional 32 bit wide dual ported RAM and several buffers to connect directly to signals from the embedding process (Figure 5).

Table 2: CIOP Interface throughput

Throughput / MBs^{-1}		single access		burst access	
		read	write	read	write
RTU	maximum	40.0	57.1	100.0	84.2
	current	26.7	44.4	48.5	59.3
PCI	PT	22.0	26.0	48.0	52.8
	optim. PT	33.0	33.0	58.6	58.6
	FIFO	33.0	33.0	52.8	52.8

As depicted the CIOP is either connected to the RTU's local bus via the glue logic mentioned above, or to the PCI bus using the so called "pass through" (PT) and FIFO modes of the S5933 PCI controller [14]. This allows different communication styles to be implemented and evaluated.

Since the PCI controller's "add–on bus" is a passive slave, a bus master state machine was implemented in programmable logic chips [15]. This keeps the interface part of the application system model small and simple, and allows us to use identical system models and configuration files for the CIOP's FPGA for both connection alternatives.

The REAR CIOP is clocked either with the RTU's 50 MHz system clock or with the 33 MHz PCI bus clock.[1] However, overhead imposed by the code generation and synthesis tools limits response times — depending on the complexity of the system model — to be ≥ 2 clock cycles.

The achievable peak data rates depend on the interface configuration: In the case of the RTU's local bus the maximum throughput is determined by the fastest possible device timing of the Galileo 64010 system controller. For our current CIOP prototype the rates are lower due to long cables between RTU and CIOP. In the PCI case the limitation comes from the master state machine controlling the add–on bus. With the current complete synchronous design a single word PCI read access takes

[1] Xilinx guarantees full PCI bus compliance for the whole XC4k–family; that implies clock frequencies up to 66 MHz and hence (theoretical) response times down to 15 ns.

6 PCI cycles whereas a 4 word burst write takes 10. Table 2 lists the calculated peak data rates for the different interface configurations "RTU local bus", "PCI pass through (PT)" (current and optimized designs) and "PCI FIFO mode". The achievable peak data rates can be computed easily by cycle counting, due to the synchronous nature of execution on the CIOP.

4 Applications

4.1 Low level control of a robot arm

As a first non time critical application example a stepper motor controller for a robot arm with 6 degrees of freedom was implemented in the CIOP. The controller consists of a programming interface (6 read/write registers), a step clock generator, 6 phase generators for the stepper motors and a multiplexer to write the phase bits cyclicly to the corresponding robot axis motors' latches within the robots power electronics. While the configurable logic block (CLB) utilization only reached 72 % of the 576 CLBs available in the XC4013E, the Xilinx partitioning placement and routing tool reported difficulties in fitting the design into the chip. In this case the routing resources limited the size of the design.

The flaw of this design is the expensive programming interface compared with the complexity of the whole task. A first improvement can be achieved by moving parts of the interface into the FPGA's IOBs. By extracting the non time critical functions, like the stepper motor speed control and even the phase generation to software, routing resources and therefore space for other I/O tasks in the CIOP could be regained.

4.2 CAN controller and monitor

A CAN bus controller and monitor system was implemented on REAR as an application which imposes a wide range of timing constraints and complexity on the implementation. CAN is a serial field bus which was originally developed for communication in vehicles, but has reached by now widespread use in the field of production automation. The CAN bus runs a masterless, message oriented bus protocol with CSMA/CA (Carrier-Sense Multiple Access/Collision Avoidance) access mode. Bus access is granted to each participant by bitwise arbitration using individual message IDs. Several cooperating error detection mechanisms guarantee fast system wide error detection and error recovery. CSMA/CA bus access, in combination with prioritization of the messages and the short data block length (max. 8 Byte) lead to a very short latency for high priority messages. The sophisticated bus protocol and data rates up to 1 MBit/s lead to an excellent performance of the CAN field bus. At the same time it demands very high quality design of components able to execute this protocol.

In our example, the REAR prototyping environment is used to implement a CAN bus monitoring and diagnosis system. Two distinct functions need to be performed by the CAN monitor: First, it has to be a fully functional *CAN bus participant*. In addition to that it needs to execute the data sampling and test signal and error generation functions necessary for *monitoring and analyzing the CAN bus*.

The individual tasks to be performed for the CAN bus participation (from now on called CAN bus controller) can be classified by an orthogonal set of attributes: The deadline of the task and complexity of the function to be performed. This is shown in Table 3 (see also Figure 2).

An analysis of the timing and complexity requirements resulting from the CAN bus protocol yields three distinct groups of tasks. At *message level*, the complexity of the tasks — message identification and message frame generation, CRC checksum generation, error protocol functions, data handling — is medium to high. The timing constraint here is identical with the length of one CAN message, $44 - 108$ μs (44 control and up to 64 data bits, at an assumed data rate of 1MHz).

The controller tasks at *bit level* — transmission of the message bits, CRC checksum error detection, bit stuffing and destuffing— need to

Table 3: CAN controller functions

Function	Deadline	Complexity
Message Level:		
Message Identification	44–108 μs	medium
Message Frame Generation	44–108 μs	medium
CRC Checksum Generation	44–108 μs	high
Error Logic	44–108 μs	high
Data Handling	44–108 μs	medium
Bit Level:		
Message Transmission Generation	1 μs	medium
CRC Error Detection	1–3 μs	medium
Bit Stuffing and Destuffing	1 μs	medium
Below Bit Level:		
Bit Timing	270 ns	low
Bitwise Arbitration	60 ns	very low

be finished in the worst case before the start of the next message bit. That results in a timing constraint of 1 μs. The complexity of these tasks is medium.

Bitwise arbitration — i. e. transmission is stopped before the next message bit if a station sending a message with higher priority ID is detected on the bus — and synchronization of the sample points while receiving the message bit stream (bit timing) are tasks with timing constraints *below bit level*. The complexity of these tasks is low to very low.

The hardest time constraints on the CAN controller are imposed by the bit timing and bitwise arbitration tasks. A worst case analysis of the timing parameters of the CAN bus protocol have resulted in deadlines of 60 ns and 270 ns, respectively. It is obvious that these requirements can only be met in hardware. The bit timing logic and transceiver control logic of the CAN controller were therefore implemented on the CIOP using the already mentioned CASE tool chain. The execution times of the arbitration mechanism and the bit synchronization were found to be 1 and 2 clock cycles respectively. So the deadlines of these two functions will be met by this implementation even if the CIOP runs with the slower 33 MHz PCI clock.

Subject of future work will be a similar analysis and implementation for the remaining CAN controller functions, which will partially be realized in software.

5 Summary and future work

We started with the analysis of different embedded high performance applications and derived a task classification regarding computation complexity and required response times. Based on this model we built a basic REAR architecture from off–the–shelf–components with appropriate processing nodes: One HPU, one RTU and one CIOP. A main purpose of REAR is to facilitate the WCET analysis on the HPU by SW–means (long computation times, interrupts disabled) and the RTU by HW–means (fast SRAM replaces 2nd–level cache).

All the necessary software tools for cross compiling, down loading, as well as the FPGA oriented part of the CASE tool chain are already in use, whereby the Statemate VHDL code generator leaves room for optimization.

The implementation of the robot arm controller has shown that our Configurable I/O Processor is capable of processing several different non trivial event driven I/O tasks. Because the CIOP is not intended as an accelerating coprocessor, but a processing unit for independent I/O tasks, the HW/SW–interface throughput is sufficient.

In the near future we plan to implement larger application examples on the REAR architecture, which take advantage of the different types of processing nodes. One challenging CIOP–task will be the high data rate I/O operations of a real–time image acquisition. While exploring the design space by moving tasks between software (RTU, HPU) and hardware (CIOP) we will gain the knowledge necessary to expand our CASE tool chain towards partitioning and generation of efficient code for the software parts.

References

[1] Klaus Gresser. An event model for deadline verification of hard real–time systems. In *Proc. Fifth Euromicro Workshop on Real Time Systems*, pages 118–123, Oulu, Finland, June 1993. IEEE.

[2] Klaus Gresser. *Echtzeitnachweis ereignisgesteuerter Realzeitsysteme*. Number 268 in Fortschrittsberichte VDI, Reihe 10. VDI–Verlag, Düsseldorf, 1993. Dissertation am Lehrstuhl für Prozeßrechner der TU München.

[3] Herbert Thielen. Automated design of distributed computer control systems with predictable timing behaviour. In J. A. de la Puente and M. G. Rodd, editors, *Proc. 12th IFAC Workshop on Distributed Computer Control Systems*, pages 47–52, Toledo, Spain, September 1994. IFAC.

[4] Kevin Jeffay. Scheduling sporadic tasks with shared resources in real–time systems. In *Proceedings of the IEEE Real–Time Systems Symposium*, pages 89–99, Phoenix, AZ, December 1992.

[5] Mani B. Srivastava and Robert W. Brodersen. Siera: A unified framework for rapid–prototyping of system–level hardware and software. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 14(6):676–693, June 1995.

[6] Massimiliano Chiodo, Paolo Giusto, Daniel Engels, Harry Hsieh, Attila Jurecska, Luciano Lavagno, Kei Suzuki, and Alberto Sangiovani-Vincentelli. A case study in computer-aided codesign of embedded systems. *Design Automation for Embedded Systems*, 1(1):51–67, January 1995.

[7] Jörg Henkel, Rolf Ernst, Wei Ye, Michael Trawny, and Thomas Benner. COSYMA: Ein System zur Hardware/Software Co–Synthese. *GME Fachbericht Mikroelektronik*, 15, 1995.

[8] Rajesh K. Gupta and Giovanni de Micheli. A co–synthesis approach to embedded system design automation. *Design Automation for Embedded Systems*, 1(1):69–120, January 1995.

[9] Raul Camposano and Jörg Wilberg. Embedded systems design. *Design Automation for Embedded Systems*, 1(1):5–50, January 1995.

[10] Pai Chou, Ross B. Ortega, and Gaetano Borriello. The chinook hardware/software co–synthesis system. In *International Symposium on System Synthesis*, Cannes, France, September 1995.

[11] Jeffrey C. Mogul and Anita Borg. The effect of context switches on cache performance. In *Proc. 4th Intern. Conference on Architectural Support for Programming Languages and Operating Systems*, pages 75–84, Santa Clara, April 1991. ACM.

[12] Yerang Hur, Yyoung Hyun Bae, Sung-Soo Lim, Sung-Kwan Kim, et al. Worst case timing analysis of RISC processors: R3000/R3010 case study. In *16th IEEE Real–Time Systems Symposium*, Pisa, Italy, December 1995.

[13] Robert Pinzinger. Speichersubsystem und flexible Prozeßanbindung für einen Rechnerknoten eines Rapid–Prototyping–Systems, 1997. Diplomarbeit (masters thesis) am Lehrstuhl für Prozeßrechner, TU München.

[14] Applied Micro Circuits Corporation, San Diego, CA. *AMCC S5933 PCI Controller Data Book*, 1996.

[15] Christian Mühlbauer. Konzeption und Implementierung einer Schnittstellenkarte mit programmierbaren Logikbausteinen zur Erweiterung einer Rapid–Prototyping Plattform, 1996. Diplomarbeit (masters thesis) am Lehrstuhl für Prozeßrechner, TU München.