A Generic Architecture for Integrated Smart Transducers*

Martin Delvai[1], Ulrike Eisenmann[2], and Wilfried Elmenreich[3]

University of Technology, Vienna, Austria,
Institut für Technische Informatik
[1]delvai@vlsivie.tuwien.ac.at, [2]eisenmann@thechilli.net,
[3]wil@vmars.tuwien.ac.at

Abstract. A smart transducer network hosts various nodes with different functionality. Our approach offers the possibility to design different smart transducer nodes as a system-on-a-chip within the same platform. Key elements are a set of code compatible processor cores which can be equipped with several extension modules. Due to the fact that all processor cores are code compatible, programs developed for one node run on all other nodes without any modification. A well-defined interface between processor cores and extension modules ensures that all modules can be used with every processor type. The applicability of the proposed approach is shown by presenting our experiences with the implementation of a smart transducer featuring the processor core and a UART extension module on an FPGA.

1 Introduction

A smart transducer is a sensor or actuator element that is integrated with a processing unit and a communication interface [1]. The processing unit transforms the raw sensor signal to a digital representation, checks and calibrates the signal, and transmits the digital information to its users via a standardized communication interface. In case of an actuator, the smart transducer accepts standardized commands and transforms these into control signals. In many cases, the smart transducer is able to locally verify the control action and provide a feedback at the transducer interface.

A demonstration of a smart transducer network is presented in the DSoS project (*Dependable Systems of Systems* (IST-1999-11585)), where smart transducers are built with commercial embedded 8-bit microcontrollers featuring integrated standard UART communication interfaces.

The miniaturization process of sensors and actuators, however, offers completely new possibilities for system designers. More and more sensor elements are microelectronic mechanical systems (MEMS) that can be integrated on the same silicon die as the associated microcontroller and the communication controller. Such an *integrated smart transducer* promises a number of advantages over a discrete design: (i) non-linear and electrically weak sensor signals can be generated, conditioned, transformed into digital form, and calibrated without any noise pickup from long external signal transmission lines [2], (ii) power consumption of the smart transducer can be reduced significantly to a level where long-life battery-powered sensors or systems powered by solar cells are possible, (iii) the size of a smart transducer will decrease to a level where the size of the

* This work was supported by the Hochschuljubiläumsstiftung der Stadt Wien via project MOSAIC (H-1147/2002).

package containing sensor, processing unit, and communication interface is insignificant compared to the size of connectors and casing, (iv) production costs for large lot sizes are considerably lower than for solutions based on a discrete design.

At the same time, a developer faces several problems when designing a smart transducer. Selecting the right microcontroller for a smart transducer is difficult since a smart transducer network can host various different nodes, where the spectrum ranges from simple sensor nodes instrumenting a contact switch up to nodes with control functions or image processing capabilities. Thus, in the majority of cases it will not be possible to select a single processor that economically covers all expected applications [3]. Moreover, if the smart transducer should provide real-time functionalities, commercial processor architectures are often not appropriate, since they are optimized for average throughput. Worst-case analysis for such systems, which is essential for real-time applications, lead to unrealistically pessimistic estimations [4]. Finally, the smart transducers, although different in processing power and interaction capabilities, should support interoperability via a standard communication interface.

It is the objective of this paper to present an architecture for integrated smart transducers that meets the above described requirements on scalability and interoperability. The paper is structured as follows: Section 2 describes the architecture of a smart transducer node in general. Section 3 describes a modular approach to build integrated smart transducers using a microprocessor core with various extension modules. Section 4 provides information about implementations of the processor core hardware, an extension module, that protects processor resources against unauthorised accesses, and a communication extension module. Section 5 gives an overview of related work in the field, while the paper is concluded in Section 6.

2 Smart Transducer Architecture

The generic smart transducer architecture introduces a two-level design approach [5] that reduces the overall complexity of a smart transducer by separating transducer-specific implementation issues from interaction issues between different smart transducers.

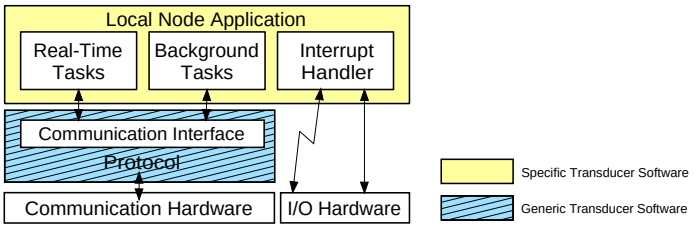

Fig. 1. Generic Smart Transducer Architecture

Figure 1 depicts the two functionalities as protocol part and local application. The protocol part instruments the network interface. Each network interface of a smart transducer must provide some standard functionalities to transmit data in a temporally deterministic manner in a standard data format, provide means for fault tolerance, and enable a smooth integration into a transducer network and its application.

For example, the smart transducer interface (STI) [6] standard specifies some services for smart transducers. It comprises a time-triggered transport service within the distributed smart transducer subsystem and a well-defined interface to a CORBA (Common Object Request Broker Architecture) environment. The key feature of the STI is the concept of an Interface File System (IFS) that contains all relevant transducer data. This IFS allows different views of a system, namely a real-time service view, a diagnostic and management view, and a configuration and planning view. The interface concept encompasses a communication model for transparent time-triggered communication. A time-triggered sensor bus will perform a periodical time-triggered communication to copy data from the IFS to the fieldbus and write received data into the IFS. Thus, the IFS is the source and sink for all communication activities. Furthermore, the IFS acts as a temporal firewall [7] that decouples the local transducer application from the communication activities. Data is transmitted in a standard UART format.

The general architecture including the protocol part applies for every smart transducer, thus implying a generic implementation approach. The realization, however, faces the following problems: Due to the differing requirements on code size and processing power for the different local applications, it is not economic to select a single processor architecture. Moreover, each smart transducer type has its own I/O hardware configuration. An architecture should support, on the one hand, the configuration of many types of I/O extensions, and, on the other hand, provide a consistent interface to these modules in order to enable software reuse. Building smart transducers on commercial hardware is possible, but due to the various microcontroller types and I/O interfaces, programming and reuse is complex and error-prone.

3 Implementation Concept

As explained in the previous section a smart transducer network comprises various nodes with different requirements in terms of processing speed, size, interfaces and so on. Due to this fact different standard microcontrollers have to be used within the same network - the programmers have to know instruction set and peculiarities of each microcontroller type in order to be able to implement, administrate, and maintain such a network. Furthermore, software pieces with the same functionality, e.g. the communication protocol, have to be implemented and tested for each microcontroller type separately. This proves to be especially difficult in the field of real time applications, where not only the correct functionality, but also the temporal behavior has to be considered. Usually, the employment of different hardware components requires a redesign of the software. To overcome these problems we designed a modular construction system, consisting of a set of processor cores and a set of different extension modules. The processor cores are all fully code compatible, but have different features in terms of computational power, required silicon area, memory-size and power consumption. Additionally, every activity within these processors is temporally predictable, which facilitates the design of real-time applications. [8]

All processor cores provide a generic interface to extension modules, which can be used to fit the microcontroller to different requirements. Figure 3 depicts the idea behind that approach.

It difficult to verify the correct functionality of a smart transducer due the fact, that not only the value domain, but also the temporal domain has to be considered. Especially the temporal aspect is critical. To evaluate the functionality of such a smart transducer in a network long simulation periods are necessary.

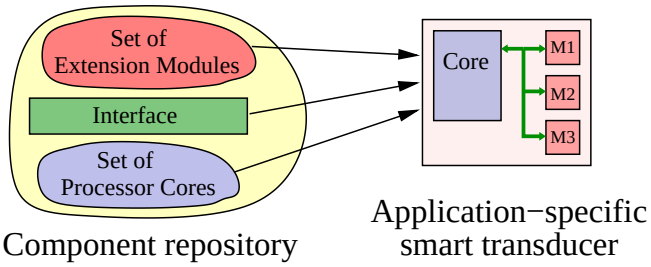

Fig. 2. Construction of an individual microcontroller

Such a simulation would require an unacceptable amount of time. Thus, it is particularly important to get prototypes of smart transducers at early development stages. The use of FPGAs is ideal for this purpose. The smart transducer can be integrated into a real network and tested over longer periods of time. Therefore, we test prototypes of the assembled network nodes on FPGA test boards. Since all employed components of our architecture are described as VHDL models it is easily possible to implement a proved system in a silicon chip.

3.1 Processor Cores

Currently, two processor cores, named SPEAR and NEEDLE [3], have been implemented by our group. While the SPEAR core is designed to support moderate arithmetic performance, NEEDLE pays attention to a compact design. A further processor core LANCE supporting high computational power is planned. All processor cores are absolutely code compatible - this means that they not only use the same instruction set, but also the same exception handling, an identical memory architecture (from a logical point of view) and the same status and configuration registers. Due to this fact the programming code can be interchanged between different processor cores without any modification.

SPEAR processor core: SPEAR features a 16 bit processor that executes instructions over a three-stage pipeline. The processor core comprises a set of 32 registers. 26 registers are general purpose registers and 6 registers have a special function: three are coupled with dedicated instructions to efficiently implement stacks and the other three registers are used to save the return address in case of a subroutine call or an exception. Data and instruction memory are both 4 kB in size. It is also possible to add up to 128 kB external instruction memory and 127 kB external data memory. The upper 1 kB of the data memory is reserved for the memory mapped extension modules. In this way an extremely simple and efficient access to these modules is provided. The SPEAR processor supports 32 exceptions, of which 16 are hardware exceptions (= interrupts) and 16 can be activated by software (= trap). The exception vector table contains the pointers to the exception service routines.

NEEDLE processor core: NEEDLE is absolutely code compatible with the SPEAR processor, but the implementation is more compact. NEEDLE has no pipeline and executes instructions within several clock cycles. Due the fact that during each clock cycle at most two memory accesses are performed, it was possible to map the entire memory architecture (instruction memory, data memory, register file and exception vector table) into a single 4 kB dual-ported memory

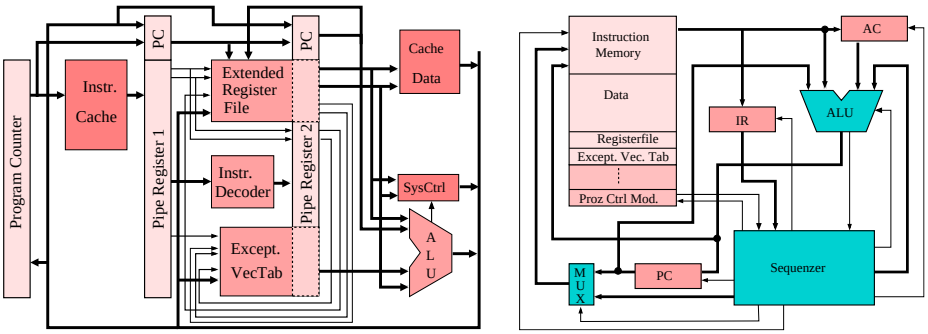

Fig. 3. Architecture of SPEAR and NEEDLE processor cores

block. Moreover, the possibility to bind up to 64 different extension modules to the data memory area still exists. The block diagram of SPEAR and NEEDLE is shown in figure 3.

3.2 Extension modules

To ensure scalability the processors are easily expandable by different extension modules, such as communication interfaces (sensors, actuators, PS/2, VGA, parallel port, USB, network interfaces, etc.) or application-specific extension modules such as a floating point unit or a protection unit. The extensions are mapped to the top address space of the data memory. For the processors the extension modules are only storage positions that can be accessed with simple load and store instructions. Therefore, from the processor's point of view it makes no difference, whether the extension is a simple sensor, actuator or a complex floating-point unit. Due to the well-defined generic interface [3] modules developed for one processor core can be used by all other cores. It is also possible to instantiate an extension module more than one time. For example, a smart transducer node can be equipped with several UART modules, having different mapping addresses in the data memory.

To illustrate the possibilities offered by the extension modules, two modules, the *protection control unit* and a customized *UART* will be described in the following. The first one extends the functionality of the processor core, the second one shows, how problems existing in commercial UART modules can be solved by a customized UART implementation.

Protection Control Module: The software that runs on a smart transducer node comprises a protocol code and an application part. As explained in section 2 the local application should not have direct access to the network bus. To guarantee that such an illegal access cannot happen, we have to ensure that only the protocol software can access the communication module. Such a protection mechanism is provided by the *protection control* extension module.

The protection control module allows assigning a *protection level* to individual data memory blocks, instruction memory blocks, registers and to extension modules. The module supports four protection levels: zero, low, high, and supervisor protection. As depicted in figure 4 the module comprises four look-up tables, which are are mapped into the data memory of the processor. Via these look-up tables a protection level to each resource can be assigned. The addresses of instruction memory, data memory, register file and extension modules are

directly connected to the input of the respective look-up table. The module controls the access by comparing the output of each look-up table (i. e., the assigned protection level) with the current protection level defined in the processor status register (i. e., protection level of the current task) and by evaluating the control signals associated to each address. When a process tries to access a resource with the current protection level being lower than the protection level of the resource, an access-violation exception is generated.

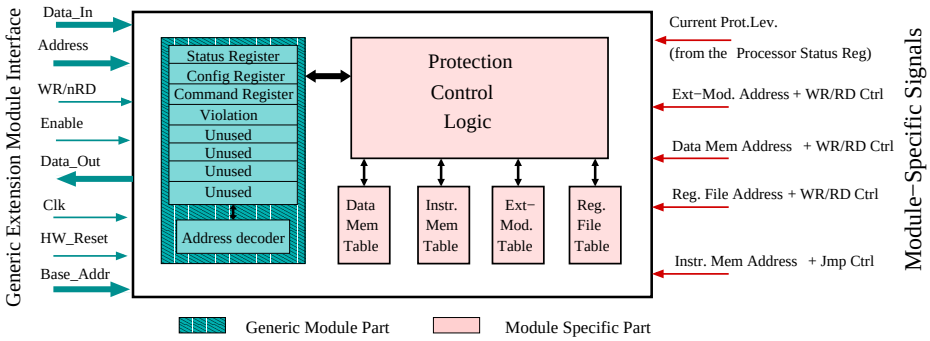

Fig. 4. Protection control module

UART Extension Module: In [9], we have examined the applicability of common UARTs for real-time communication and identified the following problems: the arithmetic error, the error due to frequency-drift and the send jitter problem. Therefore, we have developed an alternative VHDL implementation of a UART unit that resolves the intrinsic UART problems and allows the implementation of more efficient protocols, respectively the employment of cheap on-chip oscillators with large drift rates.

The UART module is able to work with an RC-oscillator that provides a frequency of 1 MHz ±50% and a frequency-drift of ±10% per second. As the baud rate is influenced by this drift it has to be continuously adjusted. To deal with the baud rate drift rate problem the UART module offers a synchronization mode. In this mode the UART can be forced to synchronize periodically, like it is necessary in fieldbus protocols such as TTP/A [10] or LIN [11]. For synchronization the UART searches for a regular bit pattern as it is outlined in Figure 5. The synchronization pattern allows the UART to determine the used baud rate.

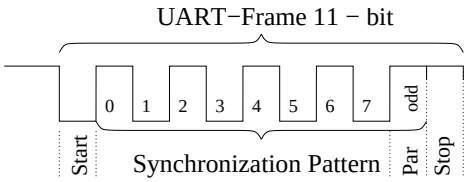

Fig. 5. Synchronization pattern

A substantial divergence from the desired transfer rate is caused by the arithmetic error in baud rate setting, i. e., the accuracy of a selected baud rate depends

heavily on the selected clock source. Therefore, it is necessary to select special crystal frequencies (e. g. $1.8432 \cdot 10^6$) to be able to use standard communication rates [9]. Our UART module uses an extrapolation mechanism that minimizes this rounding errors.

As a further problem, many commercial UARTs exhibit intrinsic delays of the sending instant of a transmission because the UART baud rate generator periodically generates potential transmission points. Thus, the start of a transmission is delayed until the next transmission point which is perceived as a send jitter by an outside user. The send jitter can be a problem for real-time communication.

Our UART module overcomes this disadvantage by a different design approach. Thus, the UART module starts with the message transmission immediately after receiving the transmit signal transmission, which almost completely eliminates the send jitter.

The bus interface contains a hardware filter that preprocesses the input signal from the bus to achieve robustness against spike interferences. Additionally, oversampling has been implemented.

Figure 6 illustrates the block diagram of the UART extension module.

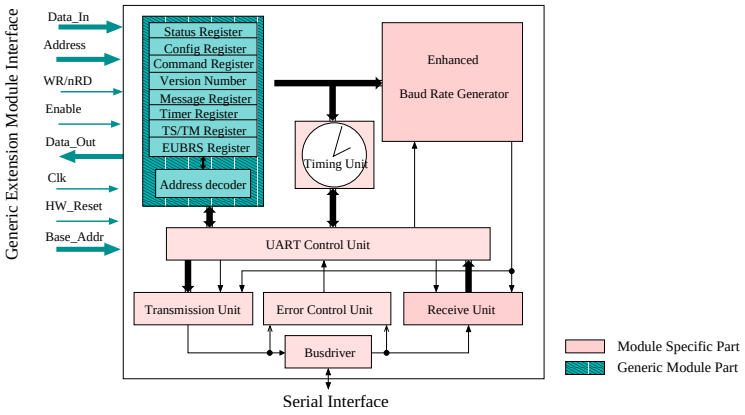

Fig. 6. Block diagram of the UART extension module

3.3 Development Tools

For software development an assembler is implemented. In the next step we will port the GNU C compiler to our processor architecture. Further a *system design tool* and a *debugger* are available to support assembling and testing of smart transducers. The system design tool provides software models of the processor cores and the extension modules. These components can be connected in order to form the desired smart transducer. The entire system could be simulated and verified. It is also possible to define own components in a high level language based on C++. In this way it is possible to evaluate a virtual hardware setup and to decide whether the real hardware should be build or not.

The debugger provides a high degree of flexibility, since it is easily adaptable to different purposes. For example it is possible to set breakpoints not only to a program counter value but also to sensor values or bus signals. Furthermore the debugger allows also the validation of real-time applications. Both, the system design tool and the debugger are controlled via the same graphical user interface.

A detailed description of the system design tool and the debugger can be found in [12, 13].

4 Experiences

We have used an Altera APEX 20KE300EQC240-1X FPGA on a Digilab prototyping board for implementing the described architecture. The modular approach has been successfully tested by composing the two processor cores with several extension modules.

The processor core properties are compared in Table 1. While SPEAR computes one instruction per clock cycle at a maximum clock frequency of 40 MHz yielding 40 MIPS (million instructions per second), NEEDLE performs one instruction within either two or three clock cycles leading to an average performance of 10 MIPS at a peak clock frequency of 25 MHz. The size of the processors is given by Logic Elements (these are the smallest hardware units in the APEX FPGA) and the instruction and data memory. Due to the different memory size of the two processor cores, the necessary silicon area for the NEEDLE core will be expected to be one half of that of the SPEAR core. The values for the power consumptions are estimates given by the Synopsys design analyzer tool.

Table 1. Comparison of the SPEAR and the NEEDLE processor

	SPEAR	NEEDLE
Maximum Speed	40 MHz/40 MIPS	25 MHz/10 MIPS
Logic Elements (LE)	1318	1010
Instr-/Data Mem	4/4 KB	2/1,92 KB
Reg File	26 GPR - 6 SPR	26 GPR - 6 SPR
Interrupts/Traps	16/16	16/16
Instructions	80	80
Power consumption	113 mW	62 mW

The size of the extension modules heavily depends on their functionality. For example, the protection control module needs 279 LE, while the UART module acounts for 699 LE.

As communication interface, we have implemented software for a time-triggered UART communication that conforms to the OMG Smart Transducer Interface standard. The implementation is supported by the UART module that, in contrast to most commercial UART units, provides temporal predictability over the send instant of a message and a minimization of the arithmetic error in baud rate setting. The improved UART module greatly reduces the implementation effort for the smart transducer protocol software.

5 Related Work

The core-based system design approach is discussed by Gupta and Zorian in [14]. Their work outlines the advantages of using predesigned, preverified, silicon circuit modules as building blocks of large and complex applications on a single silicon die. ALTERA[1] offers a set of processor cores for embedded applications

[1] http://www.altera.com/products/devices/nios/nio-index.html

called Nios® following a similar idea as our proposed approach. The main difference for both approaches lies in its real-time capabilities. WCET analysis for code segments running on the Nios processor version 2.0 or higher is difficult, since the execution time of a single instruction depends on the preceding and the following instruction, the operands used, how recently operands were modified and other additional factors. In contrast our processor cores provide a fully temporal predictable behavior, which make them more suitable for to be used in real-time smart transducer networks.

Related work on sensor integration with VLSI circuitry can be found throughout the literature. Main research focuses on wide bandgap semiconductor materials [15], thin film sensors [16], and MEMS devices [17]. These approaches are related to the work presented in this paper as they outline possibilities for further on-chip I/O modules. The examined literature on smart sensor technologies, however, usually neglects the integration of a transducer with an appropriate communication network interface.

Besides the OMG STI, there are some other communication standards for smart transducers. Many fieldbus protocols, such as Controller Area Network, Local Area Network, Local Interconnect Network, Profibus, Foundation Fieldbus, WorldFip, and Interbus also provide possible solutions for the communication interface. The main differences between these approaches are in real-time features and implementation complexity. We have chosen the OMG STI since it provides hard real-time capabilities while having a low implementation complexity. In general, our architecture supports the easy adaption to a different communication interface by exchanging or adding communication I/O modules.

Another related standard is the IEEE 1451 smart transducer standard, which is not another fieldbus protocol but can be treated in the same way in our case, since IEEE 1451 specifies a 10-wire transducer-independent interface [18], which could be implemented as a communication extension module in our architecture.

6 Conclusion

This paper presented an architecture for the implementation of integrated smart transducers, i. e., a sensor or actuator element, a microcontroller and a network interface on a single silicon die.

The key element of our architecture is a set of fully code compatible microprocessor cores, with a well-specified interface to hardware extension modules that are synthesized on the same semiconductor chip. Due to the modular approach of the proposed architecture, the system is open to various external extension modules such as physical sensors/actuators or network communication modules.

Currently we have implemented two different microprocessor cores and several extension modules. The microprocessor cores are compatible at register and machine language level, so that software can be easily reused. The two extension modules are designed to support a smart transducer implementation. The improved UART module reduces the implementation effort for the smart transducer protocol software, while the protection unit protects resources like communication interfaces from unauthorized access.

References

1. W. Elmenreich and S. Pitzek. Smart transducers – principles, communications, and configuration. In *Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems (INES'03)*, volume 2, pages 510–515, Assuit – Luxor, Egypt, March 2003.
2. P. Dierauer and B. Woolever. Understanding smart devices. *Industrial Computing*, pages 47–50, 1998.
3. M. Delvai, U. Eisenmann, and W. Huber. Modular construction system for embedded real-time applications. In *Tagungsband of Austrochip 2002*, Vienna, Austria, 2002.
4. P. Puschner and A. Burns. A review of worst-case execution-time analysis. *Journal of Real-Time Systems*, 18(2/3):115–128, May 2000.
5. S. Poledna, H. Angelow, M. Glück, M. Pisecky, I. Smaili, G. Stöger, C. Tanzer, and G. Kroiss. TTP two level design approach: Tool support for composable fault-tolerant real-time systems. *SAE World Congress 2000, Detroit, MI, USA*, March 2000.
6. Object Management Group (OMG). *Smart Transducers Interface Final Adopted Specification*, August 2002. Available at http://www.omg.org as document ptc/2002-10-02.
7. H. Kopetz and R. Nossal. Temporal firewalls in large distributed real-time systems. *Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '97)*, pages 310–315, 1997.
8. M. Delvai, W. Huber, P. Puschner, and A. Steininger. Processor support for temporal predictability - The SPEAR design example. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal, July 2003.
9. W. Elmenreich and M. Delvai. Time-triggered communication with UARTs. In *Proceedings of the 4th IEEE International Workshop on Factory Communication Systems (WFCS'02)*, Västerås, Sweden, August 2002.
10. H. Kopetz et al. Specification of the TTP/A protocol. Technical report, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, March 2000. Available at http://www.ttpforum.org.
11. Audi AG, BMW AG, DaimlerChrysler AG, Motorola Inc. Volcano Communication Technologies AB, Volkswagen AG, and Volvo Car Corporation. LIN specification and LIN press announcement. SAE World Congress Detroit, http://www.lin-subbus.org, 1999.
12. M. Delvai, M. Jankela, and A. Steininger. Towards virtual prototyping of embedded computer systems. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI'03)*, Orlando, FL, USA, July 2003.
13. M. Delvai, C. El Salloum, and A. Steininger. A generic real-time debugger architecture. In *Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI'03)*, Orlando, FL, USA, July 2003.
14. R. K. Gupta and Y. Zorian. Introducing core-based system design. *IEEE Design & Test of Computers*, 14(4):15–25, Oct.-Dec. 1997.
15. B. W. Licznerski, K. Nitsch, and H. Teterycz. Polycrystalline wide bandgap materials in sensor technology. In *Abstract Book of the 3rd International Conference on Novel Applications of Wide Bandgap Layers*, pages 32–35, Poland, 2001.
16. S. M. Vaezi-Nejad. Advanced sensors scene in europe. In *Proceedings of the IEEE Instrumentation and Measurement Technology Conference*, volume 2, pages 926–931, Ottawa, Canada, May 1997.
17. G. K. Fedder. Structured design of integrated MEMS. In *Proceedings of the 12th IEEE International Conference on Micro Electro Mechanical Systems*, pages 1–8, January 1999.
18. Institute of Electrical and Electronics Engineers, Inc. *IEEE Std 1451.2-1997, Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Micro-processor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, September 1997.