ENHANCING FPGA ROBUSTNESS VIA GENERIC MONITORING IP CORES

Alexander Biedermann, Thorsten Piper, Lars Patzina, Sven Patzina,
Sorin A. Huss, Andy Schürr, and Neeraj Suri
CASED (Center for Advanced Security Research Darmstadt),
Mornewegstraße 32,
64293 Darmstadt, Germany
{*alexander.biedermann|thorsten.piper|lars.patzina|sven.patzina|sorin.huss|andy.schuerr|neeraj.suri*}*@cased.de*

Keywords: System Monitoring; Embedded HW/SW Design; FPGA Monitoring

Abstract: Today, state of the art technology allows a very dense integration of embedded HW/SW designs. As a consequence, more errors are introduced in these circuits that have to be observed during run-time. Adding monitors to a design enables the recognition of and the reaction to these threats, but, usually, monitors have to be developed for every individual FPGA design. Our approach provides generic IP cores that permit the monitoring of arbitrary hardware modules. Furthermore, by providing a central monitoring module, statements about the behaviour of the entire system can be made.

1 INTRODUCTION

Nowadays, embedded systems spread over nearly every application domain. With advancement of technology the integration density of Integrated Circuits (ICs) increases and with it, further sources of errors are introduced. Such systems co-operating in networks as communicating control units or sensors are exposed to threats from the outside world. These can be the manipulation of sensors or injection of malicious messages that can cause severe security and safety risks. To reduce these, an appropriate technique as monitoring has to be applied to make specific statements about the system behavior. Suitable monitors can be realized as a wrapper around a single component of the network or as separate instance in the network. Often these communicating systems have real-time constraints that do not allow high delays introduced by a monitoring approach. Therefore, a realization of these monitors completely in hardware or in a combination of hardware and software is needed to achieve required performance. Though, realizing such monitoring functionalities in an FPGA design needs the manual implementation of these observing structures. The design and realization of such monitors for different systems are time-consuming and repetitive tasks. However, the structure of such monitoring wrappers and components are very similar to each other, which demands a generic solution.

Our contributions are:

- a framework for instrumenting arbitrary IP cores with wrappers in an FPGA design
- a central monitoring core for more complex monitoring functions that aggregates and coordinates the wrappers
- a predefined repository for generic and predefined monitoring functions
- possibility to extend the repository by self-developed monitoring modules

Therefore, we extend the Embedded Development Kit (EDK) developed by Xilinx, the market leader in reconfigurable logic, by generic monitoring IP cores. The extensible toolbox includes predefined logical monitoring blocks that can be tailored for special proposes. These are used to add monitor functions to existing or newly developed FPGA designs.

The remainder of this paper is organized as follows. In Section 2 we take a look at related work. Afterwards, in Section 3 we present the architecture of our framework and show its applicability in a temperature measurement scenario in Section 4. Section 5 concludes our approach.

2 RELATED WORK

Monitors are powerful and versatile tools to gain insight on the internal states and signals of a system. Monitors can furthermore be applied to ensure the correct operation of a component or to detect and react on faulty component behavior. Monitors are widely applied in embedded systems, as there are many areas of application throughout the lifecycle of a product, ranging from design time to run-time. During design time, embedded monitors aid in debugging and profiling a system or parts thereof, whereas at run-time, they provide performance measures and ensure operability of the system.

Embedded logic analyzers (ELA) such as Xilinx's ChipScope Pro (Xilinx, 2010) or Altera's SignalTap II (Altera, 2010) are the simplest form of monitors in terms of complexity. ELAs provide direct access to internal signals of a system and, therefore, constitute an essential tool during test and verification of a design, as only few components offer externally accessible interfaces. Their use is limited to scenarios where the direct exposure of a signal path without further processing is desired or sufficient.

Debugging tasks that require triggering on specific events or sequential patterns depend on a more sophisticated monitoring solution than ELAs can provide. To allow for more complex scenarios, (Penttinen et al., 2006) presents a method to monitor the internal signals of FPGA circuits by using an embedded microprocessor as central monitoring instance. Their setup is able to account for timing constraints and does not suffer from slowdowns in case of many or complex input patterns, like HDL simulators do.

Another approach is taken in (Cheng et al., 2010), where the authors propose a run-time RTL debugging methodology for FPGA-assisted co-simulation. By instrumenting the design under test (DUT) with a wrapper, parts of the design simulation are executed transparently on an FPGA. Furthermore, their approach provides internal node probing and, thus, achieves full observability of the DUT.

Profiling and performance monitoring (PM) (Sprunt, 2002) constitutes another application area of hardware monitors. PM is commonly implemented by performance event (PE) detectors and PE counters. PE detectors trigger on certain events, such as distinct program characteristics, memory access, pipeline stalls, branch predictions or resource utilization and increment their corresponding PE counter. The implementations of existing performance monitoring solutions such as (DeVille et al., 2005), (Lancaster et al., 2010) and (Tong and Khalid, 2008) are often highly application dependent and offer no or low reuseability. With the framework for generic monitoring IP cores presented in this paper, the instrumentation of HW components with monitoring functionality is highly automated. By using pre-defined monitoring blocks from the frameworks's library or by extending the library with user-supplied blocks, code reuse for common monitoring tasks is easily achieved.

Based on similar considerations, the authors of (Schulz et al., 2005) propose Owl, a framework to pervasively deploy programmable monitoring elements throughout a system. Owl is built on the architectural principle of programmable capsules, which are comparable to the wrapper notation of our approach, and analysis modules that provide functionality within the capsules. Programmable capsules are realized as reconfigurable and programmable logic in an FPGA and are used to integrate hardware monitors at potential event sources. Although their framework is suited for generic application scenarios, the authors' main focus is on profiling. They propose memory access logging, memory access histograms and dynamic pattern recognition and reduction as possible use cases. One of the major differences to our approach is the absence of a central monitoring core (CMC) that provides a facility to aggregate systemwide monitoring events and to coordinate systemwide reaction patterns during run-time.

Apart from implementing existing monitoring solutions, our proposal aims at extending the area of application of HW monitors in embedded systems. As related work shows, HW monitors are traditionally employed for tasks such as debugging, profiling and performance measurement. In addition to this, we aim to implement well-established software concepts of the dependability, safety and security domains, especially those which are commonly implemented by component wrappers. In this context, the proposed framework can therefore be seen as first step to deliver an enabling platform upon which further research and experiments will be conducted.

Currently, we consider several future application scenarios. The first one is to implement fault-containment wrappers (Saridakis, 2003) for IP cores, in order to limit a fault's effect to the wrapped component and prevent its propagation to other parts of the system. We furthermore consider to use monitors for the safe implementation of component isolation in mixed-criticality designs (Pellizzoni et al., 2009) and also investigate to provide monitors that offer reaction triggers for reconfiguration in self-adaptive autonomic systems (Santambrogio, 2009).

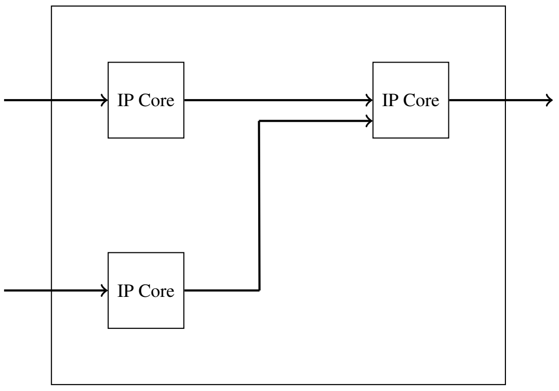


Figure 1: An exemplary FPGA design.

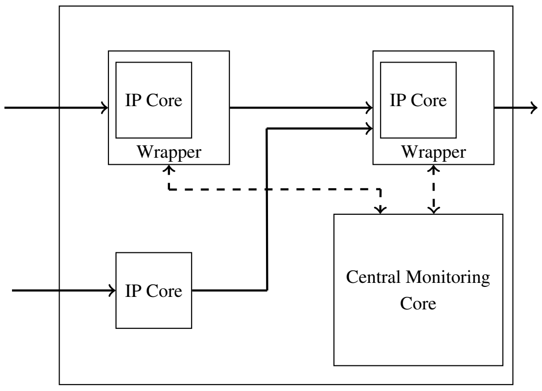


Figure 2: The same design with Monitoring Modules inserted.

3 MONITORING ARCHITECTURE

The approach presented in this paper delivers a framework for automatic integration of monitoring functionalities into existing HW/SW designs. A smooth integration into the existing design flow for reconfigurable devices from Xilinx is provided. A simple exemplary design consisting of three IP cores is shown in figure 1. To add monitoring functionalities, predefined monitoring functions from a repository can be easily inserted into a design. Thus, monitoring of an IP core of a system is achieved by wrapping a Monitoring Module Wrapper (MMW) around it. Furthermore, a so-called Central Monitoring Core (CMC) coordinates all connected Monitoring Module Wrappers. It allows for coordinating all MMWs and for monitoring the entire system, if every IP core in the design is wrapped by a MMW. After adding these monitoring components to the exemplary system, the result can be seen in figure 2. MMWs and the CMC communicate over a dedicated communication structure. The dedicated communication is displayed by dashed lines in this figure. In the following, we describe the components of our monitoring framework.

3.1 Monitoring Interconnect Structure

All monitoring-related communication is independent from the communication structure of the original system. Therefore, besides delays caused by monitoring functions, normal system communication of modules is not affected by the monitoring functionalities. The communication of the entire monitoring system relies on the Fast Simplex Link (FSL) (Rosinger, 2004). The FSL is well-suited for several reasons:

- It has a lightweight protocol with very fast data handling: in each clock cycle, a data packet can be sent.
- It supports asynchronous communication between different clock regions.
- Besides many existing hardware modules, the Microblaze processor provided by Xilinx supports FSL communication via 16 independent FSL ports.
- It has a FIFO buffer of adjustable depth to catch stalls caused, for example, by temporary bursts of monitored data.

For demonstration purposes, we restricted ourselves for now to monitor only modules, which also employ FSL interfaces. It is possible to support other interfaces and bus standards.

3.2 Monitoring Module Wrapper

To observe the status of a module, a Monitoring Module Wrapper (MMW) can be placed around a module. Such a wrapper for a simple IP core with two inputs, i and j, and two outputs, o and p, is depicted in figure 3. Each of the inputs and outputs of a module can be selected to be monitored by the MMW. Ports which are not monitored are left untouched. Monitored ports are fed into the input switch of the MMW. For example, in figure 3, input i and its adjacent output p are monitored. The Monitoring Module Wrapper allows to monitor and evaluate an output port or relations between input and output ports of modules. For the last of these options, the input switch waits as long as the output delivered from the monitored IP core matches its corresponding input. Until then, the corresponding input is buffered in the input switch. However, to match a value from an input with its corresponding value on an output port, the designer has to denote the processing time of the IP core in the configuration of the input switch. In further research, the time

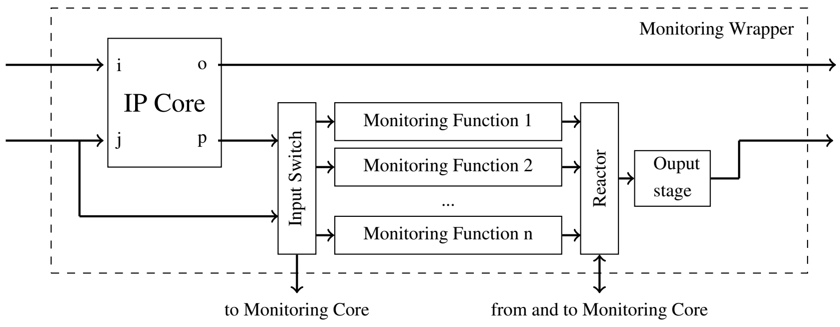


Figure 3: An IP-Core wrapped by the Monitoring Module Wrapper. Input *j* and output *p* are monitored. Synchronization between these ports is handled in the Input Switch.

needed to buffer an input will be calculated automatically if a constant processing time is assumed. The monitored values are now sent to the processing stage of the wrapper as well as to the Central Monitoring Core which is explained in Section 3.3. The processing stage contains monitoring functions, which analyze the monitored ports. Monitoring functions in the MMW are intended to quickly deliver statements about the monitored values. For example, a monitoring function may evaluate, if monitored values are within a predefined range of values. Several of such predefined and configurable monitoring functions are held ready in a repository as VHDL-written modules. The designer may chose, which monitoring functions he wants to add to the current wrapper. This is done by editing a configuration file. If a malfunction or deviation of a module is detected by a monitoring function, the monitored output port may be altered by the reactor within the MMW. The designer may configure the reactor of the MMW to define which monitoring function is prioritized, if more than one of them detects a malfunction or deviation. If and in which way an output will be altered is defined in the corresponding monitoring function. In some cases of detected erroneous behavior, it may be appropriate to send a predefined default value instead of leaving the wrong data untouched. Furthermore, rather than altering an output value, an error message may be sent to the CMC via the Reactor.

3.3 Central Monitoring Core

The Central Monitoring Core is a configurable IP core of our monitoring framework designed to evaluate the state of a group of monitored modules or of the entire system. By aggregating information gathered by the independent MMWs, the CMCs is able to combine these data to make statements about the system's status. Reactions to deviations may be triggered by the CMC and may be sent to the corresponding MMWs. Furthermore, the Central Monitoring Core is intended to run more complex monitoring functions than in MMWs. Mainly, if several monitored values from the past have to be evaluated, a monitoring function within the MMW is not well-suited, since they are intended to deliver statements quickly. Therefore, Each MMW is automatically connected to the CMC, delivering a set of inputs and outputs of the monitored module that have to be monitored. The designer can define in each MMW, which ports he wants to monitor in the CMC. As for the MMW, the designer may choose, which monitoring functions are included into the CMC. In contrast to the MMW, monitoring functions in the CMC may use inputs and outputs from more than one monitored module. As a consequence, statements about a group of modules or even the entire system are possible. The CMC consists of configurable central monitoring functions, a switch interface, a log file memory block and an optional soft core processor. A CMC is depicted in figure 4.

3.3.1 Switch Interface

The CMC communicates with the MMWs via FSL interfaces. A switching module coordinates communication between wrapped modules and the CMC's monitoring functions. The designer may configure, which inputs and outputs of the wrapped module are sent to the central monitoring functions added into the CMC. If a monitoring function of the CMC triggers an output manipulation for a monitored port, the intended value is sent via FSL to the reactor of the corresponding wrapped module. Output manipulations triggered by the CMC are intended to correct long-term deviations of a module. Here, non-blocking writes are used to send altered output data from the CMC to the output switch of a wrapper. Since monitoring functions in the CMC may take a longer time to deliver results, the output switch of the MMW is,

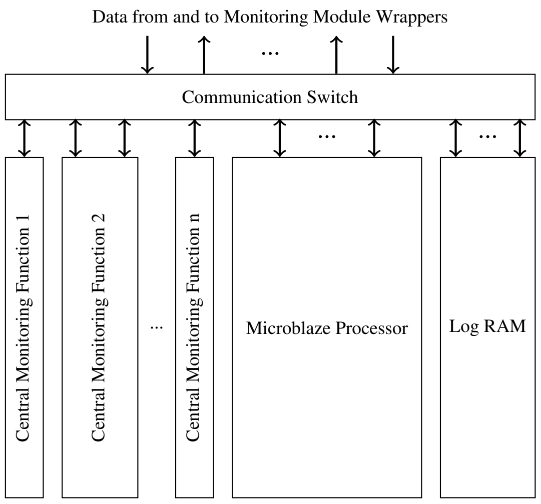

Data from and to Monitoring Module Wrappers
...
Communication Switch
...
...
Central Monitoring Function 1
Central Monitoring Function 2
...
Central Monitoring Function n
Microblaze Processor
Log RAM

Figure 4: The Central Monitoring Core and its connections to the MMWs.

therefore, not blocked by waiting for results from the CMC.

3.3.2 Log File Memory Block

Since some central monitoring functions might need monitored data, which were captured in the past, a dedicated memory saves monitored values. For each central monitoring function in the CMC, the designer may adjust, if and when monitored values have to be stored in the log file memory block. Therefore, the size of the log file memory block is adjustable. Central monitoring functions, as well as the optional Microblaze soft core processor, access the log file memory via the communication switch. For the log file memory block, the on-board BlockRAM of an FPGA is used.

3.3.3 Microblaze Soft Core Processor

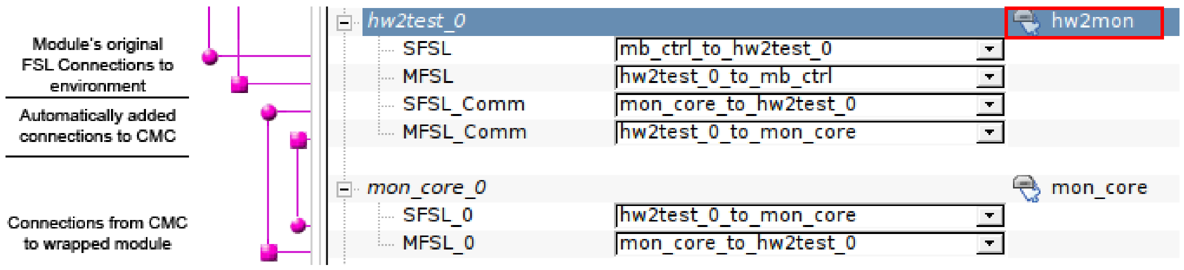
Writing combinatorial monitoring functions in VHDL often delivers fast results. However, especially in complex monitoring scenarios, the use of a high-level programming language is desirable to describe monitoring functions. Therefore, within the CMC, an optional, C-programmable Microblaze soft core processor is available. The designer may either implement several monitoring functions by himself or use some of the predefined C-written monitoring functions available in a repository. As for VHDL-written monitoring functions, the Microblaze uses FSL as communication interface for monitored values. For each port of a wrapped module that has to be evaluated using the Microblaze, a dedicated FSL connection is connected between the communication switch and the Microblaze. The communication switch sends the values from monitored module ports to the corresponding FSL ports of the Microblaze. Up to 16 independent ports may be evaluated on a Microblaze.

3.4 Integration into the Xilinx Design Flow

In our approach, we provide a smooth integration into the Xilinx design flow. Our framework is aimed at being used in combination with the Xilinx Embedded Delevopmenet Kit (EDK). The EDK is mainly used to create reconfigurable HW/SW designs by instantiating and connecting IP cores. To add monitoring functionalities into the design, the designer has to configure a script file written in Python. It defines, which existing modules of an EDK design have to be wrapped and which monitoring functions will be added to the MMWs and the CMC. In Section 4, we show for an exemplary design, how the script is configured to add monitoring functionalities. By executing this script, the configured Monitoring Module Wrappers are then wrapped around the selected IP cores. As a result, in Xilinx EDK, the modules are replaced by a wrapped instance of themselves. Figure 5 shows a wrapped module with automatically created connections to the CMC. Note that the IP type – marked by a red border on both figures – of a wrapped IP core changes to ”hw2mon”. The module’s connection to its environment persists. Additionally, connections to the Central Monitoring Core and the Core itself are automatically created. Then, the normal design flow, such as bitstream generation may be executed.

4 EVALUATION AND DISCUSSION

To demonstrate the integration of our generic monitoring approach into existing designs, we implemented a simple temperature sensor station. The design which will result from the following process is shown in figure 6. Three independent sensors based on thermistors deliver values of resistance. In three VHDL-written hardware modules on an FPGA, these raw values are normalized based on the thermistors’ characteristics. Output of these modules are discrete temperature values. Afterwards, these values are then sent to a Microblaze soft core processor via FSL connections. It visualizes the current temperature data as well as the average of the three sensors on a display. We now want to monitor the nor-


Module's original FSL Connections to environment
Automatically added connections to CMC
Connections from CMC to wrapped module
hw2test_0
hw2mon
mon_core_0
mon_core

SFSL	mb_ctrl_to_hw2test_0
MFSL	hw2test_0_to_mb_ctrl
SFSL_Comm	mon_core_to_hw2test_0
MFSL_Comm	hw2test_0_to_mon_core
SFSL_0	hw2test_0_to_mon_core
MFSL_0	mon_core_to_hw2test_0

Figure 5: The EDK showing a wrapped IP core with automatically created connections to the automatically added CMC.

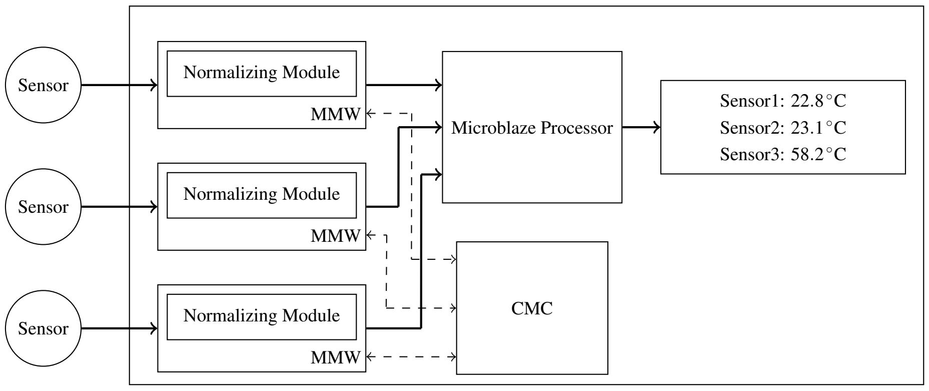


Figure 6: An FPGA design with three temperature sensors, three mapped normalizing modules, a Microblaze with an attached display, and the CMC.

malizing modules to detect possibly abnormal temperature values. Therefore, we add Monitoring Module Wrappers around the normalizing VHDL modules by configuring and executing a Python script. Such an exemplary configuration is shown in figure 1. In the configuration file, we define to wrap the IP core *normalizingModule* which converts raw values into temeprature values. Thus, two existing monitoring functions, *mmw_value_range*, and *mmw_threshold* are inserted. They test, if the monitored port delivers data which are within a range of values or lie below a defined threshold. These values are passed via generics into the monitoring functions. In the configuration file, these monitoring functions are then bound to the corresponding port. Furthermore, the CMC is configured to include the monitoring function *cmc_tendency*. It monitors, if the tendency of the monitored data changes, e.g. if the monitored temperature data fall after a period of rising. At last, a Microblaze is added and connected to the wrapped module. Further monitoring functions written in C can be added to the Microblaze. After executing the Python script, the EDK has all of the configured monitoring functionalities inserted. For each MMW around the normalizing modules, monitoring a pair of input and output values consumes 106 LUTs and 208 slice registers on a Virtex-5 FPGA. Resource consumption of the CMC without monitoring functions added, and without the optional Microblaze added is 86 LUTs and 69 Registers. A Microblaze, if used within the CMC, consumes 916 LUTs and 583 Registers. For each central monitoring function added and for each MMW connected to the CMC, the resource consumption will rise. On the one hand, this is due to the resource consumption of the central monitoring functions. On the other hand, this is due to the generic structure of the communication switch of the CMC. Its complexity rises with each new monitoring module that has to be connected to the CMC.

As seen in figure 3, the input and output switches as well as the monitoring functionalities add a delay to monitored ports. Therefore, our approach might not be suited for systems with harsh timing constraints. In this case, the constant delay caused by the MMW might be considered by the designer of an IP core in advance. However, is possible to eliminate the delay caused by the monitoring functionalities: Here, monitored output ports are forked. One branch is immediately sent out of the module. The other branch is fed into the input switch of the MMW. Indeed, cor-

Listing 1: Configuration script to add monitoring functionality to an existing design.

```
## Monitoring Configuration File.
# project path
project: ../sensorstation/

#Name of entity and of file of the VHDL-module which has to be wrapped
[normalizingModule]
# name of the pcore-module after wrapping
output: normalizingModule_monitored
#corresponding VHDL file
file: normalizingModule.vhd

# Automatically replace module with MMW-wrapped instance
# if false, the new monitored module can be added manually into the design
replace: true

#Microprocessor Peripheral Definition/Peripheral Analyze Order File needed by EDK
#(found automatically if project path is given)
mpd: normalizingModule/data/normalizingModule_v2_1_0.mpd
pao: normalizingModule/data/normalizingModule_v2_1_0.pao

#Ports of wrapped module to monitor by MMW
#format: <port>_portID_<direction>
port_0_out: temp_out

#Monitoring Functions to add
#format: <monitor>_portID_monitorID [generic1,generic2,...]
monitor_0_0: mmw_value_range -20,40
monitor_0_1: mmw_threshold 50

## Central Monitoring Core Configuration File
[core]

# which CMC Monitor Functions to include
monitor_0: cmc_tendency

# bind each CMC Monitoring Function to one or several MMWs
cmc_tendency_0: normalizingModule

# assign a microblaze to the core
microblaze_name: cmc_mb
# comma separated list of wrapped modules which will be connected to the microblaze
microblaze_entities: normalizingModule
%\caption{Configuration script to add monitoring functionality to an existing system.}
```

recting a detected deviation would not be possible any more, since the anomalous value was already sent to its adjacent modules via the first branch. We are currently analyzing, in which cases and scenarios the delay caused by our Monitoring Module Wrapper is a reasonable price to pay to gain the ability to alter output values. As said in Section 3.1, we restrict ourselves to monitor modules, which communicate via FSL. By using the blocking read and write structure of the FSL interface for modules we attenuate the negative side effects caused by delays, because modules adjacent to the monitored module will wait as long as the observed output is available.

5 CONCLUSION

In this paper, a novel design method to add monitoring functionality to HW/SW designs is presented. Monitoring functionality can be easily added by automatically wrapping modules and by choosing suitable monitoring functions from a repository. A Central Monitoring Core coordinates monitored modules and is able to make statements about the entire system's status. Complex monitoring functions can be implemented in C via the Microblaze soft core processor optionally included in the Central Monitoring Core. Since the monitoring system uses a communication structure independent of the communication structure of the original system, the monitoring system smoothly integrates into existing designs. Apart from a delay caused by the evaluation of the monitoring functions in the wrappers, the system's communication is not affected. The application scenario in 4 shows that our approach is suitable to be used in existing HW/SW systems. Resource consumption depends on the amount of Module Wrappers employed, the confiugration of the CMC with or without Microblaze and the number of Monitoring Functions added. Further research includes the use of the monitoring system in distributed environments. There, several interconnected embedded systems are monitored by independent monitoring systems. The Central Monitoring Cores of these systems may then communicate with each other. As a result, not only the status of a single system, but the status of an entire network of embedded systems can be gained. Possible fields of application are distributed sensor networks as well as security-related sensor systems.

ACKNOWLEDGEMENTS

This work was supported by CASED (http://www.cased.de).

REFERENCES

Altera (2010). *Design Debugging Using the SignalTap II Logic Analyzer.*

Cheng, X., Ruan, A., Liao, Y., Li, P., and Huang, H. (2010). A run-time rtl debugging methodology for fpga-based co-simulation. In *Communications, Circuits and Systems (ICCCAS), 2010 International Conference on*, pages 891 –895.

DeVille, R., Troxel, I., and George, A. (2005). Performance monitoring for run-time management of reconfigurable devices. *Engineering of Reconfigurable Systems and Algorithms (ERSA).*

Lancaster, J., Buhler, J., and Chamberlain, R. (2010). Efficient runtime performance monitoring of FPGA-based applications. In *SOC Conference, 2009. SOCC 2009. IEEE International*, pages 23–28. IEEE.

Pellizzoni, R., Meredith, P., Nam, M., Sun, M., Caccamo, M., and Sha, L. (2009). Handling mixed-criticality in soc-based real-time embedded systems. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 235–244. ACM.

Penttinen, A., Jastrzebski, R., Pollanen, R., and Pyrhonen, O. (2006). Run-Time Debugging and Monitoring of FPGA Circuits Using Embedded Microprocessor. In *IEEE Design and Diagnostics of Electronic Circuits and systems*, pages 147 –148.

Rosinger, H. (2004). Connecting customized IP to the MicroBlaze soft processor using the Fast Simplex Link (FSL) channel. *Xilinx Application Note.*

Santambrogio, M. (2009). From Reconfigurable Architectures to Self-Adaptive Autonomic Systems. In *International Conference on Computational Science and Engineering*, pages 926–931. IEEE.

Saridakis, T. (2003). Design patterns for fault containment. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003), Germany.*

Schulz, M., White, B. S., McKee, S. A., Lee, H.-H. S., and Jeitner, J. (2005). Owl: next generation system monitoring. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 116–124, New York, NY, USA. ACM.

Sprunt, B. (2002). The basics of performance-monitoring hardware. *IEEE Micro*, 22(4):64–71.

Tong, J. and Khalid, M. (2008). Profiling Tools for FPGA-Based Embedded Systems: Survey and Quantitative Comparison. *Journal of Computers*, 3(6):1.

Xilinx (2010). *ChipScope Pro 12.3 Software and Cores - User Guide.*