FPGA-Based Realization of Self-Optimizing Drive-Controllers

C. Paiz, J. Hagemeyer, C. Pohl, M. Porrmann, U. Rückert
System and Circuit Technology Group
Heinz Nixdorf Institute
University of Paderborn
Fürstenallee 11, Paderborn, Germany
Email: {paiz,jenze,pohl,porrmann,rueckert}@hni.upb.de

B. Schulz, W. Peters, J. Böcker
Institute for Power Electronics
and Electrical Drives
University of Paderborn
Warburger Str. 100, Paderborn, Germany
Email: {schulz,peters,boecker}@lea.upb.de

***Abstract*—An FPGA (Field Programmable Gate Array) implementation and suitable power electronics can lead to a fast torque response in motion drive applications. However, when the controller parameters or its structure have to be adapted to internal and external varying conditions, e.g., when a self-optimizing control system is pursued, a static implementation might not lead to the best utilization of reconfigurable resources. This contribution outlines the implementation of a self-optimizing system composed of several possible hardware and software realizations of controllers for a permanent magnet servo motor. How well a specific controller realization is suited to the current situation is evaluated based on control quality and realization effort (i.e., CPU time, reconfigurable area). A System-on-Chip architecture is presented, which enables an on-line exchange of FPGA- and CPU-based realizations of controllers to optimize resource utilization and control quality. It is shown that by using dynamic hardware reconfiguration, such self-optimizing controller can be implemented based on FPGA technology. Furthermore, the design-flow including self-developed tools is outlined. Experimental results show that the proposed scheme works satisfactory.**

I. Introduction

Field Programmable Gate Array (FPGA) technology has become an attractive alternative to implement digital control systems, because it offers an interesting trade-off between performance, design effort, and cost for various application fields, e.g., industrial controllers [1], or embedded applications [2]. If a control system is designed to optimize itself (e.g., parameter or structure adaptation) based on internal and external objectives, then a whole family of controllers might be required to cover all possible states of the controlled system. Such control systems are known as self-optimizing systems [5]. When realizing such a system in reconfigurable hardware, an implementation where the configuration of the FPGA does not change during operation (static realization), leads to a high resource-overhead, since all possible control variations have to be placed concurrently, even when they are not required. To overcome this, dynamic and partial hardware reconfiguration can be used, e.g., to load only the suitable controller for the current situation of the system.

In the next section a self-optimizing scenario is described. The system consist of FPGA- and CPU-based controllers, a motor, and power electronics, as described in section III. Section IV shows the FPGA-based System-on-Chip architecture. The design-flow including on- and off-line verification and the automatic design integration into the flow is outlined in section V. Measurement and simulation results are presented and discussed in section VI. Finally, conclusions are given in section VII.

II. Self-Optimization Scenario

For this contribution, a complex mechatronic system composed by many sub-tasks is considered. The computational hardware is shared among all sub-tasks, and must be understood as having limited resources, e.g., memory, CPU-time, or FPGA area. The drive-control sub-task is composed of various controllers, and different realizations of those controllers (e.g., CPU- or FPGA-based), which consequently have different computational requirements, and control characteristics. In a self-optimizing system, a control algorithm may be understood as an optimal solution for the current internal and external objectives of the system. Therefore, to each possible situation of the system, there is a drive controller, and correspondingly an FPGA- or CPU-based implementation of that controller, which represents a solution in that situation. Thus, to realize controllers that compete with other sub-tasks of the mechatronic systems to access the limited computational resources, the optimal operational condition of each controller and their possible realizations should be known. The common abilities and some realization aspects of motor controllers are well known (cf. [3]) and shown in Tab. I. With respect to the drive application, a concurrent FPGA-based realization of all required control algorithms would enable an adaptive control system, as presented in [4]. However, the amount of computational resources that have to be allocated to that sub-task from the mechatronic system would be too high. By introducing partial run-time reconfiguration the resource allocation can be improved, and thus the mechatronic system can assign free resources to other sub-tasks.

According to the definition of self-optimization [5], the decision to switch between different control algorithms or different implementations of those algorithms has to be taken in three steps:

TABLE I
ABILITIES AND REALIZATION ASPECTS OF MOTOR CONTROLLERS.

No.	Control Algorithm	Situation		Complexity	
		revolution speed	load torque behavior	computing time per cycle	required sample rate
1	FOC (P-Controllers)	low	constant	low	slow
2	FOC (PI-Controllers)	low	constant	low	slow
3	Back EMF compensation	medium	constant	medium	medium
4	current decoupling	high	fluctu-ating	high	high
5	Direct Torque Control	very high	fluctu-ating	low	high

1) "Analysis of the current situation":
   By determining the kind and amount of available resources (Memory, CPU-Time and FPGA-area), and the current situation of the controlled system.
2) "Determination of objectives":
   By distinguishing the optimal solution for the total mechatronic system according to the drive application as well as to the cost-benefit ratio for the control switching. The characteristics of the available controllers (cf. Tab. I) and their implementation (cf. Tab. II) are considered in this step.
3) "Adaptation of the system behavior":
   Accomplishing control switching (if required). This step has a direct influence on the available computational resources and the control quality.

The cyclic repetition of these three steps satisfies the self-optimizing-framework [5]. This contribution focuses on the control drive sub-task, without considering a concrete mechatronic system or other sub-tasks. Different realizations of well-known control structures are used to explore controller switching between several kinds of implementations.

III. DRIVE-CONTROL STRUCTURES

To focus on the capability of FPGA architectures as well as the switching strategy for controllers it is fundamental to select well analyzed control algorithms.

Fig. 1 shows the block diagram of a torque controller, which regulates the system by controlling a vector of current components: the d-current and the q-current. On the one hand, the d-current has to be controlled to zero in order to avoid energy losses in the motor. On the other hand, the q-current has to be controlled to adjust the torque driving the motor. All controller structures used in this contribution are based on a Field Oriented Control (FOC) scheme, which has been presented in literature by several authors, e.g., [6], and [7]. In the elementary control structure of an FOC-scheme the output of a PI-controller is directly the output of the controller. The medium scaled structure (FOC-EMF) contains a feed-forward for Back-EMF compensation. As such, the dynamics of the control loop is improved for speed changes. The large scaled control structure (FOC-EMF-DeC) features an additional decoupling of the currents to improve the behavior in the case of high-dynamic load torque changes.

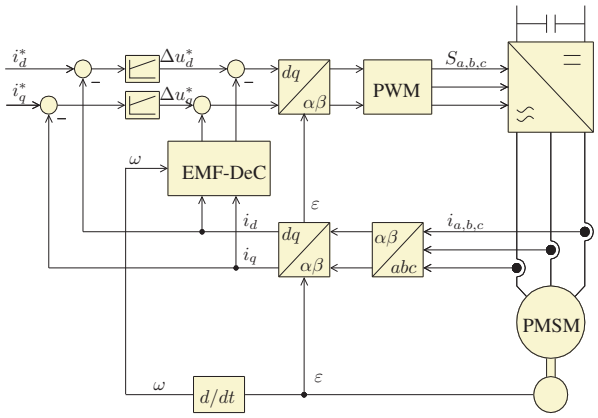

Fig. 1. Field orientated control structure with Back-EMF compensation and decoupling of currents.

A. Test-Bed description

The test bed used for the presented studies is shown in Fig. 2. The test bed consists of a rapid prototyping system (RAPTOR system [8]), power electronic for a permanent magnetic motor, which is the Equipment Under Test (EUT), and a load machine.

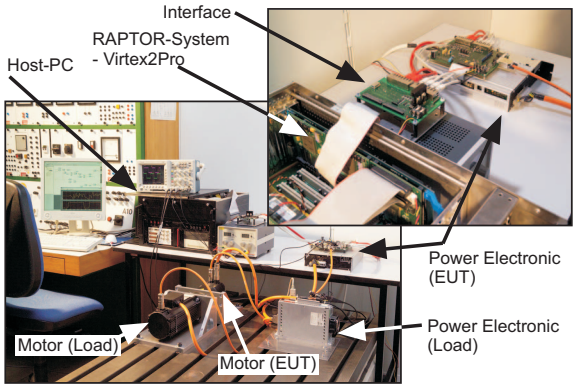

Fig. 2. Test bed for FPGA-based controllers.

A simplified schematic of the information processing system and its connection to the EUT is shown in Fig. 3. Power electronics and computation hardware are connected through a fully isolating digital interface board for sensor and actuator signals. Firing signals for the power electronic are created in the FPGA system, and are transmitted as digital signals for the switches.

The ADC uses a delta-sigma-modulator, which allows that the quantization as well as the sampling rate of the current sensor signals are scaled by an optimized decimation filter. The utilization of sensor signals for current and position in the computation hardware is supported by the power electronic system. This test bed allows emulating many different drive applications, such as speed, position and torque control. The special capability of this test bed is the on-line reconfiguration

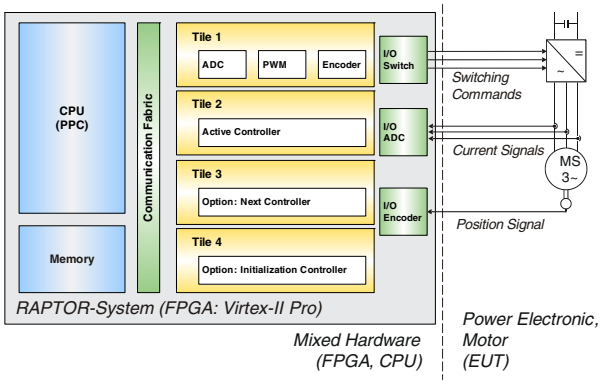

Fig. 3. Structure of modules for dynamic reconfiguration of FPGA- and CPU-based motor controllers.

of the drive controllers. This is not restricted to only FPGA-base controllers; the exchange of CPU- and FPGA-based realizations is also supported. This feature requires a flexible underlying information processing system, not only because different realizations of the controllers are supported, but also because the information flows from sensors and to actuators have to be reconfigurable at run-time. A System on Chip (SoC) designed for this purpose is presented in section IV.

The presented FPGA-based controllers were realized using the Xilinx System Generator [9], and following the design-flow presented in section V. FPGA resources for the different control structures and the hardware interface used for hardware-in-the-loop (HiL) simulations and measurements are given in Tab . II.

The realization of CPU-based controllers is commonplace in industrial drive applications. Furthermore, the theoretical and practical aspects of CPU-based drive control reconfiguration can also be found in literature [10], [11], and [3]. Therefore, the realization of such standard CPU-based controllers is not presented. However, the dynamic reconfiguration of FPGA-based controllers, and the switching from an FPGA- to a CPU-based controller is a new step in drive controllers.

A comparison of the execution-time of FPGA- and CPU-realized controllers on our SoC architecture is presented in Fig. 4. The PowerPC works with a clock frequency of 300 MHz, whereas the reconfigurable Tiles have a clock frequency of 30 MHz. The PWM-Carrier is depicted to illustrate the timing constraint of the controllers (i.e., control cycle). The used Delta-Sigma-ADC is realized using regular sampling. As

TABLE II
RESOURCES OF IMPLEMENTED FPGA-BASED CONTROLLERS.

Structur:	Hardware Interface	FOC		FOC-Back EMF		FOC-EMF-DeC	
FPGA Resources:		Ctrl	Init	Ctrl	Init	Ctrl	Init
Slices	1273	360	194	453	283	671	497
FlipFlops	1052	153	74	172	91	272	187
LUTs	1525	605	340	767	502	1154	889
BRAMs	1	4	0	4	0	4	0
MULTs	1	0	0	0	0	2	2

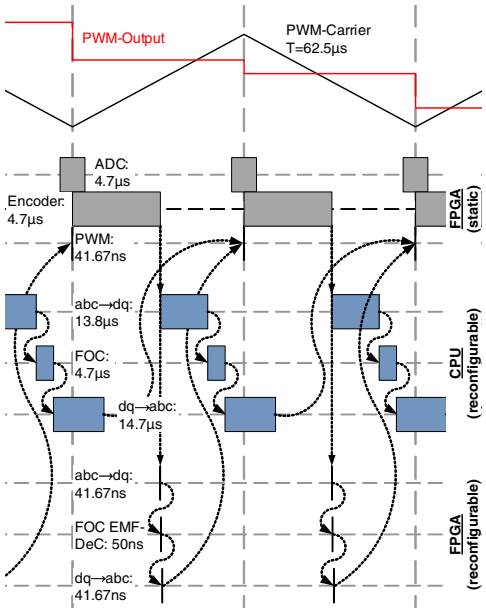

Fig. 4. Comparison of the execution-time of FPGA- and CPU-based motor controller realizations.

can be seen, the timing of the encoder is dominated by the serial data transfer and synchronized to the ADC. The outlined timing of static part of the PWM supports the displacement for the zero-voltage vector of the set-voltages (Zero Sequence Signal). Even though the precise timing depends on the actual clock-rate, it can be noticed that the execution time of the CPU realization is longer than a single control cycle, causing the controller to have a time delay of one sample period. The FPGA realization is two orders of magnitude faster than the CPU realization, and has no significant time-delay. This speed-up comes from the concurrent utilization of several processing elements (cf. Table II), in contrast to the serial realization of the CPU-based controller. The low execution-time of the FPGA-realization enables the implementation of more complex control schemes (e.g., a speed-adaptive PWM-period can be easily implemented).

IV. SYSTEM-ON-CHIP ARCHITECTURE

To realize the previously described self-optimizing control strategy, a SoC architecture based on the self-developed RAPTOR prototyping system [8] has been implemented. The architecture was realized on one daughter board of the RAPTOR system, which features a Xilinx Virtex-II Pro FPGA (XC2VP30). The architecture is composed of an embedded PowerPC processor (PPC) connected to dynamically reconfigurable resources (Tile 1 to 4), and to other hardware components described in this section. Furthermore, a processor local bus (PLB) allows communication to the local bus (LB) of the RAPTOR system, and from there to the host PC, through the PCI bus, as depicted in Fig. 6.

The architecture can be divided into static and dynamic components. Tile 1 to 4 are fixed slots of equal size, which can be dynamically reconfigured, i.e., a new partial bitstream can

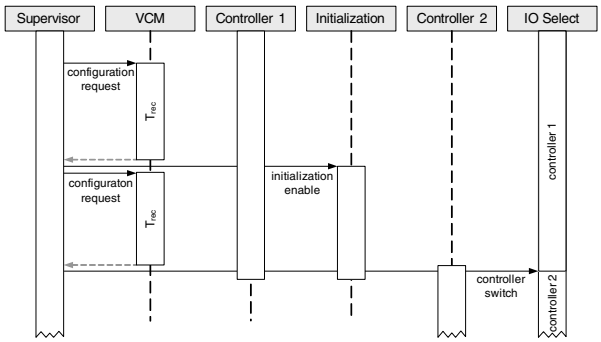

Fig. 5. Reconfiguration flow. The total reconfiguration time (T_{rec}) is the same for any of the four Tiles.

be loaded at run-time while the rest of the system remains operative. These slots are used to implement controllers or signal conditioning blocks, since these elements are exchanged according to the current state of the plant and the current objective of the system (cf. section II).

The reconfiguration of any of the Tiles is carried out by the Virtex Configuration Manager (VCM) [12]. A program running on the PPC can initiate the reconfiguration, indicating the memory space from the external SDRAM where the partial bitstream ought to be copied. The destination Tile is embedded in the partial bitstream. When a reconfiguration is requested, the VCM initiates DMA (Direct Memory Access) transfers from the SDRAM controller, loads the desired partial bitstream to the target Tile by accessing the ICAP (Internal Configuration Acces Port), and sends an interrupt to the PPC when done (cf. Fig. 5). This process last about 4,38 ms, which represents several control cycles. To overcome this, an initialization routing is used to calculates the initial states of the new-loaded controller. A supervising program, running in the PPC, is in charge of monitoring system activity and triggering the reconfiguration of any of the Tiles (cf. section II).

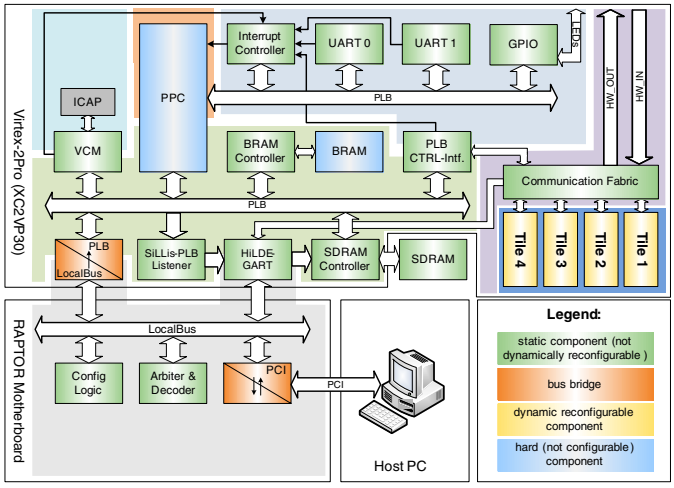

Fig. 6. Schematic of the system architecture implemented on the RAPTOR system.

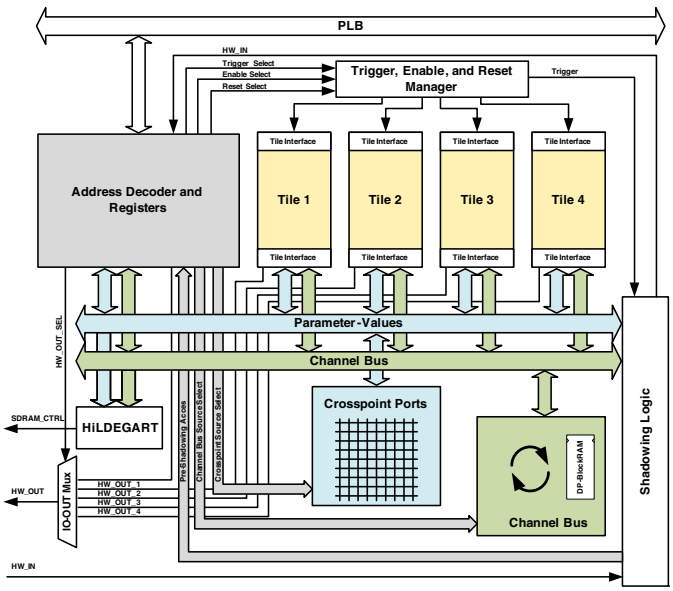

Fig. 7. Simplified schematic of the communication fabric.

The architecture incorporates a flexible communication system, enabling data transmission between static and dynamic components (e.g., between Tiles and the PPC), as well as between internal components and external components (e.g., between controllers and the plant). All Tiles are connected through a multiplexer to the external components, a select signal controlled by the PPC defines one of the Tiles as current output, which can be changed as required at run-time (cf. Fig. 5). Furthermore, each Tile has four 16-bit I/O ports (called Crosspoint ports, cf. Fig. 7), and up to 64 16-bit wide IO ports connected to the Channel Bus. Crosspoint ports are suited for data-streaming between Tiles (e.g., initialization of a new-loaded controller from another Tile). Channel Bus communication is slower, and best suited for parameter exchange.

The communication fabric is highly configurable, allowing the PPC, to set the source for the 16-bit Crosspoint ports, Channel Bus and the outputs (HW_OUT, cf. Fig. 7). Any of the Tiles or the PPC can be the source of any of the Crosspoint ports of the Tiles; this configuration can be changed at run-time. For system monitoring, the PPC has access to all values and Channel Bus vectors. Furthermore, as the PPC has access to all signals of the system, its also possible to implement a controller completely in software without utilizing some of the reconfigurable slots.

This feature is specially important to the realization of the presented self-optimizing scheme, because based on this flexibility a controller running in the PPC can be exchanged by other FPGA-based controller, placed in any of the Tiles, which can be initialized by a dedicated design placed in any other Tile or even the PPC.

V. Design-Flow

The design of a dynamically reconfigurable control system requires special methods and tools such as off-line and on-

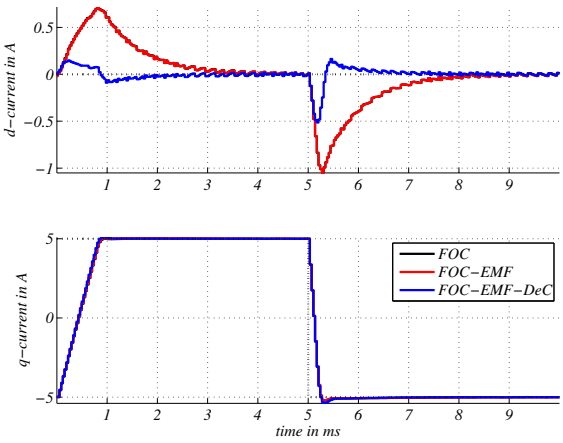

Fig. 8. q-current step response at 3000 rpm (HiL: Controller at FPGA, Motor in Simulation)

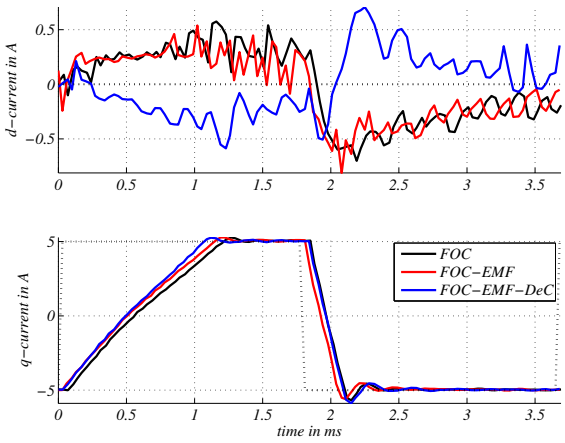

Fig. 9. q-current step response at 3000 rpm (Test bed: Controller at FPGA, Motor as EUT).

line hardware-in-the-loop (HiL) simulations. Furthermore, the flow must support the automatic integration of a controller into those HiL frameworks, and the automatic generation of the partial bitstreams. In this section, the tool-flow supporting the design process of self-optimizing controllers is presented.

For off-line FPGA-in-the-Loop simulation HiLDE (Hardware-in-the-Loop Development Environment) has been developed [13]. HiLDE is a cycle-accurate testing framework for performing FPGA-in-the-Loop simulations, where the focus is on the functional verification of the design, using a simulated environment. The Design Under Test (DUT) is automatically encapsulated into a hardware wrapper, to enable the connection to and synchronization with a simulation tool such as Matlab/Simulink or ModelSim.

A logical step after performing a cycle-accurate functional design verification is to realize a real-time verification of the DUT. For this purpose, HiLDEGART (HiLDE for Guided Active Real-Time Test) was developed [13]. Our approach allows monitoring and parameterizing a running controller in real-time.

Generating the hardware wrappers for HiLDE and HiLDEGART is an application for vMAGIC [13]. The starting point of the flow is a VHDL file containing the DUT's entity definition, which is then analyzed by vMAGIC. vMAGIC automatically generates DUT-specific wrappers and configuration files for HiLDE or HiLDEGART. An example of a HiLDE simulation is shown in Fig. 8, showing a verification function of controller. Measurements with the real EUT, done using HiLDEGART, are presented in Fig. 9, validating the results of the HiL simulation. Both results show a proper control behavior of all three presented control algorithm.

The partial bitstreams that are required to configure the target FPGA during run-time have to be generated separately. This means that for every controller and for all possible controller positions (e.g., target Tile) a partial bitstream has to be generated. These steps are realized automatically by using our Integrated Design Flow for Rconfigurable Architectures (INDRA) [12].

VI. Control Switching Validation

For validation of a proper switching between controllers, a HiL simulation of such control exchange is presented in Fig. 10. The motor is first controlled with a FOC (No. 2 in Tab. I), and at time-point zero the control is switched to a FOC-EMF-DeC (No. 4 in Tab. I). As can be seen, switching was done without disturbing the controlled currents. To enable this bump-less control switching, a proper initialization of the internal state of the controller (e.g., integral initial state) is required, as discussed in [14]. Without such an initialization the controlled currents show a disturbance at the time of the switching, as can be observed in Fig. 11. In this figure measurements of a control switching with the EUT are shown, using the same controllers as in Fig. 10, but without initialization.

Fig. 12 shows measurements of a control switching between a CPU-based FOC algorithm using a P-controller (Tab. I, No. 1), and a FPGA-based FOC with a PI-controller on the EUT. The amount of noise of the current is defined by the selection of the control algorithm, its realization, and external

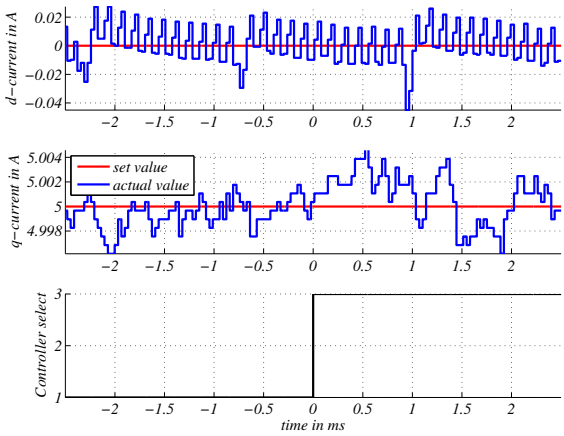

Fig. 10. Controller switching from FOC to FOC-EMF-Dec at 3000 rpm (HiL: Controller at FPGA, Motor in Simulation).

perturbations. On the one hand, the P-controller used for the measurements shown in Fig. 12 produces low noise, but produces also a steady state error. On the other hand, the PI-controller has a better steady state response, but requires more resources for its implementation. These measurements and simulations results show that the presented concept works satisfactory.

VII. Conclusions

The realization of an FPGA-based self-optimizing motion controller was presented. This approach allows for the adaptation of parameters and structure of controllers. Furthermore, not only the control algorithm, but also its realization and the execution platform (FPGA or embedded CPU) can be dynamically changed. Basis of this realization is a System-on-Chip (SoC) architecture that enables the use of dynamic hardware reconfiguration, and the run-time adaptation of the communication infrastructure.

It was shown that switching between different FPGA-based realizations and from an FPGA- to a CPU-based realization (and vice versa) can be done. Furthermore, considering the short execution times of FPGA-based controllers, and the possibility to still use a CPU-based controller, allows the adaptation of the control system, not only regarding the controlled system, but also regarding the available resources of the SoC architecture and how they are to be used by other systems. This empowers the control system to react to situations far beyond the classic approaches.

This contribution also presented a tool-flow supporting the design of FPGA-based controllers, making such a task easier to non-hardware engineers, and less error prone to every user. Finally, measurements from experiments with the presented test-bed show that the proposed scheme works satisfactory, motivating further research in this area. The combination of a real-time operating system with the presented architecture is being investigated, as well as the improvement of the HiL framework.

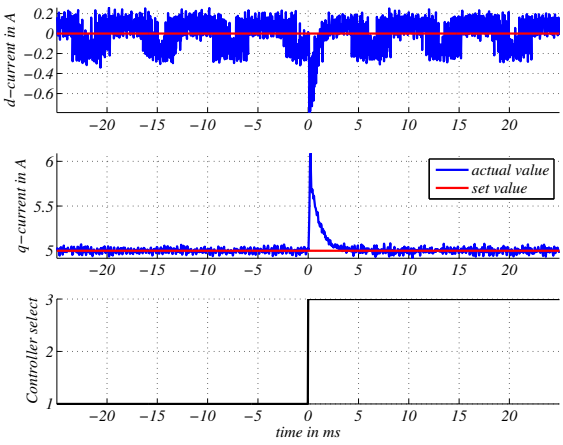

Fig. 11. Controller switching from FOC to FOC-EMF-Dec at 3000 rpm (Test bed: Controller at FPGA, Motor as EUT).

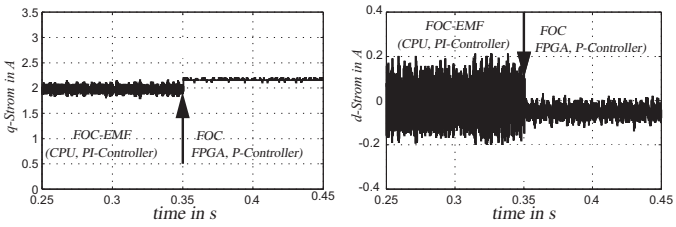

Fig. 12. Controller switching from a CPU- and an FPGA-based realizations at 3000 rpm (Test bed: Controller at CPU and FPGA, Motor as EUT).

Acknowledgment

This work was supported by the Collaborative Research Center 614 - Self-Optimizing Concepts and Structures in Mechanical Engineering - University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

References

[1] E. Monmasson and M. Cirstea, "FPGA Design Methodology for Industrial Control Systems—A Review," *Industrial Electronics, IEEE Transactions on*, vol. 54, no. 4, pp. 1824–1842, Aug. 2007.

[2] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu, "An overview of reconfigurable hardware in embedded systems," *EURASIP J. Embedded Syst.*, vol. 2006, no. 1, pp. 13–13, 2006.

[3] E. Monmasson, B. Robyns, E. Mendes, and B. De Fornel, "Dynamic reconfiguration of control and estimation algorithms for induction motor drives," in *Industrial Electronics, 2002. ISIE 2002. Proceedings of the 2002 IEEE International Symposium on*, vol. 3, 2002, pp. 828–833 vol.3.

[4] S. Mathapati and J. Böcker, "Implementation of Dynamically Reconfigurable Control Structures on a Single FPGA Platform," *12th European Power Electronics and Adjustable Speed Drives Conference, Aalborg, Denmark*, 2007.

[5] J. Böcker, B. Schulz, T. Knoke, and N. Fröhleke, "Self-Optimization as a Framework for Advanced Control Systems," *Int. Electronics Conference (IECON), November 2006, Paris, Frankreich*, 2006.

[6] F. Blaschke, "The Principle of Field Orientation as Applied to the New Transvektor Closed-Loop Control System for Rotating-Field Machines," in *Siemens Review*, vol. 34, May 1972, pp. 217–220.

[7] K.-H. Bayer, H. Waldmann, and M. Weibelzahl, "Field-Oriented Closed-Loop Control of A Synchronous Machine With the New Transvektor Control System," in *Siemens Review*, vol. 39, 1972, pp. 220–223.

[8] M. Porrmann, J. Hagemeyer, J. Romoth, and M. Strugholtz, "Rapid prototyping of next-generation multiprocessor socs," in *In Proceedings of Semiconductor Conference Dresden, SCD 2009*, Dresden, Germany,, 2009.

[9] *System Generator Users Guide*, 10th ed., Xilinx, Inc., San Jose, USA, 2008.

[10] E. Ho and P. Sen, "A microcontroller-based induction motor drive system using variable structure strategy with decoupling," *Industrial Electronics, IEEE Transactions on*, vol. 37, no. 3, pp. 227–235, Jun 1990.

[11] A. Khambadkone and J. Holtz, "Vector-controlled induction motor drive with a self-commissioningscheme," *Industrial Electronics, IEEE Transactions on*, vol. 38, no. 5, pp. 322–327, Oct 1991.

[12] J. Hagemeyer, B. Kettelhoit, M. Koester, and M. Porrmann, "Design of Homogeneous Communication Infrastructures for Partially Reconfigurable FPGAs," in *Proceedings of the International C onference on Engineering of Reconfigurable Systems and Algorithms (ERSA '07)*, Las Vegas, USA, 2007.

[13] C. Paiz, C. Pohl, and M. Porrmann, "Hardware-in-the-Loop Simulations for FPGA-Based Digital Control Design," in *Informatics in Control, Automation and Robotics*, vol. 3. Springer-Verlag, 2008.

[14] B. Schulz, C. Paiz, J. Hagemeyer, S. Mathapati, M. Porrmann, and J. Böcker, "Run-Time Reconfiguration of FPGA-Based Drive Controllers," in *12th European Conference on Power Electronics and Applications, September 2-5, 2007*, Aalborg, Denmark, September 2007.