

## PC-based Low Latency Controller for Dynamic Mechatronic Systems

*Folker Schwesinger, Gunter Krapf, Thomas Fröhlich*

Technische Universität Ilmenau, Institute for Process Measurement and Sensor Technology

### ABSTRACT

This paper presents a processing platform for digital control systems based on personal computers (PC). The proposed system aims to provide low-latency data acquisition, processing and output, a both dynamic and easy to use prototyping platform during control system development as well as a flexible and adaptable architecture for productive control operation. A field-programmable gate array (FPGA) is used as an interface module to effectively connect external, serial peripheral interface (SPI)-compatible components to the PC system. External SPI input data is converted into and output data is extracted from a transparent PCI Express (PCIe) data stream, which is accessible from within the control software executed on the PC processor cores. A real-time Linux operating system (OS) is used in order to achieve deterministic, low-latency data processing tasks implemented in software on the PC.

The publication addresses the motivation of designing such a system for controlling electromagnetic force-compensating (EMFC) balances. System requirements and used components are presented, and detailed hard- and software concepts are introduced. Achieved measurement results are shown and discussed.

*Index Terms* – digital control system, low-latency data processing, FPGA, PCI Express, SPI, real-time Linux, EMFC

### 1. INTRODUCTION

In countless time-critical applications like feedback control loops, the implementation of functionalities in analog electronics is and will be state of the art. However, it is difficult to implement complex controller structures in analog circuitry. In addition, these analog controllers have a significant disadvantage concerning flexibility and adaptability. That is why the trend in science and technology has been increased towards all-digital control systems. Against this background it can be observed that especially the digital control of complex dynamic mechatronic systems like electromagnetic force-compensated (EMFC) balances gains increasing importance.

Common digital controllers base on an analog to digital converter (ADC), a digital processing unit and a digital to analog converter (DAC). Here the processing unit usually consists of a microcontroller, a digital signal processor (DSP) or an FPGA. But also the usage of PCs comes to mind as a processing platform in digital control, both during development and productive operation. Especially the “PC-in-the-Loop” concept has various advantages compared to common signal processor or FPGA solutions. During control system development, PC-based platforms provide a very powerful as well as a highly flexible and adaptable prototyping system, because control algorithms can be implemented in software running on the PC. Hence a control system designer is able to implement control algorithms in standard programming languages like Assembly, C or even higher-level languages. During this process no special knowledge about specific processing units is needed. Thus adaptations

or complete translations of the control algorithm are not necessary that allow execution on special processing units or other target platforms. Performing control algorithm implementation tasks on higher abstraction layers promises to result in shorter development times, which effectively lead to a shorter time to market in contrast to classical control system design approaches.

Once in production use PC-based architectures also promise essential advantages over traditional microcontroller, DSP or FPGA solutions: Here limitations regarding available processing resources, like memory and computing power are negligible. Moreover the highly flexible software implementation of the control algorithm allows for very easy and fast changes and algorithm updates (e.g. modifications of filter coefficients in digital filters). Thus time-consuming reprogramming of processing chips and cost-intensive replacement of hardware modules are avoided.

Despite the promising advantages of employing PC-based system architectures in digital control development and usage, several challenges need to be addressed in order to guarantee both real-time capable and low-latency operation. Common PCs equipped with conventional interfaces and multitasking operating systems show significant discontinuities and latencies in data transfer and processing speed. However, a defined runtime performance and fast response time are mandatory for controllers in terms of the stable control of physical systems.

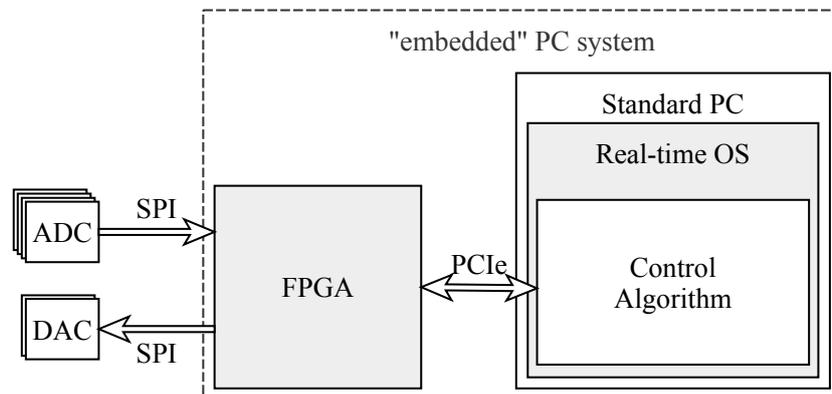


Figure 1: Block diagram of the developed embedded PC system

Within this context a PC system was designed that overcomes the described problems by using a real-time Linux operating system and an FPGA-based interface for the direct low-level and low-latency access of multiple ADC and DAC circuits (see figure 1). A transparent application programming interface (API) provides an abstraction layer that provides access to control system data streams and functionalities while hiding hardware-specific details from the application.

Here the usage of commercially available standard component and free open-source software tools was prioritized. Using the example of a controlled EMFC balance the functional capabilities of this system were analyzed and optimized.

## 2. SYSTEM ARCHITECTURE

The system architecture of the proposed PC-based control system has a logical structure consisting of different abstraction layers. According to the ISO/OSI reference model [1], here the complexity of the system is distributed among different layers as shown in figure 2.

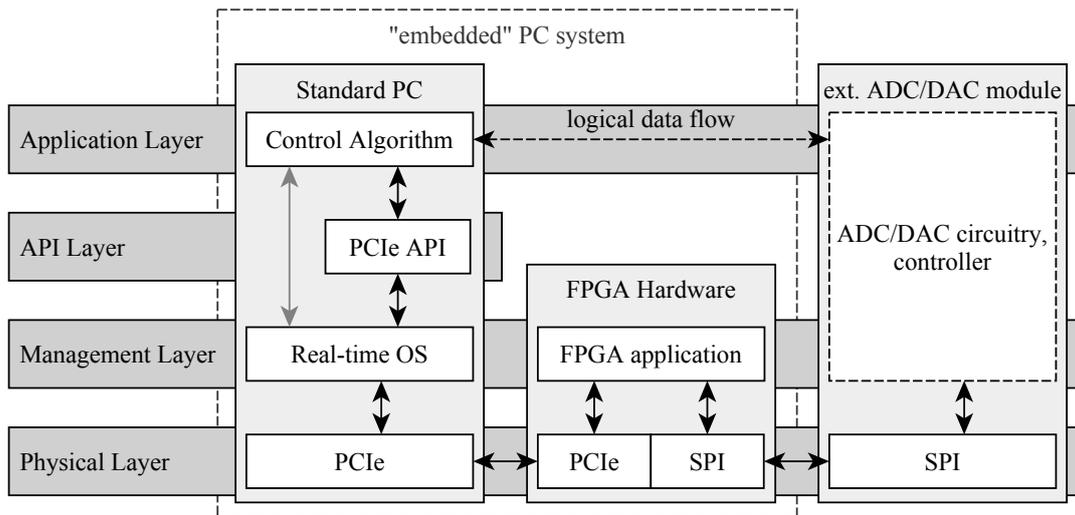


Figure 2: Layer model of the developed embedded PC system

The architecture of the proposed control system is divided into four layers. Software programs that implement control algorithms belong to the application layer and use services and functionalities provided by the underlying API layer and management layer. As the name suggests the PCIe API developed during the course of our research is resident in the API layer. Operating system functions, FPGA module functions as well as circuitry of external components are encapsulated in the management layer. It is conceivable to divide the management layer into several sub-layers to allow for a more subtle granularity. However one single abstraction layer was chosen at this point because most of the functions and services belonging to the management layer are provided by third-party components. These components form atomic modules that were not modified or adapted. Only the FPGA application is an exception. It was especially designed to fit the needs of the system. Over-the-wire data transmissions take place on the physical layer. From the user (and control algorithm developer) perspective the logical data flow occurs on the application layer between the different communication endpoints, namely the control algorithm and external ADC/DAC modules which provide and expect control data. Thus operating system-, FPGA- or external component-specific details are effectively hidden from the user.

## 2.1 System overview

As shown in figure 1 and 2 the control system presented in this paper consists of a standard PC that is equipped with a PCIe-based FPGA board. Table 1 lists the relevant system specifications of the PC setup. The deployment of a PCIe-enabled FPGA card as an interface component between the standard PC and external circuitry allows for low-latency, parallel data transmissions which cannot be achieved with other standard PC interfaces like RS232, Universal Serial Bus (USB) or Ethernet. The CESYS PCIeV4Base [2] FPGA board was chosen over several alternative PCIe-capable FPGA boards. Only this board is equipped with an integrated PLX PCIe controller enabling straightforward PCIe communications between FPGA and host PC. Furthermore CESYS guarantees generic Linux OS support and provides valuable sample applications including source code for software and FPGA applications. Characteristic FPGA board properties can be obtained from table 2.

Processor type and model	Intel Core i7 CPU 950, 3 GHz
Main memory	12 GB RAM
Operating system	Ubuntu Linux 12.04 (64bit), Kernel 3.8.9
Real-time extension	CONFIG PREEMPT RT Patch 3.8.9-rt4

Table 1: Embedded PC system specifications

External components like ADCs and DACs are connected to the FPGA via the PIB64IO [3] daughterboard which provides 64 user configurable general purpose input/output (GPIO) pins. Some of these IO pins are used as data and control signal lines for the SPI interfaces of the connected conversion modules. The FPGA implements an application that serially reads incoming data samples from multiple connected ADCs. Once an input data set arrives in the FPGA registers it is forwarded to the PCIe interface enabling the control algorithm running on the PC to fetch the data samples by issuing PCIe API read calls.

After software-based data processing is finished, output data is written back to the FPGA. Again, this is done by using appropriate PCIe API calls. The output data stream is taken by the FPGA from the PCIe interface and converted to a serial, SPI-compatible data stream, which is forwarded to the connected DACs.

According to the described functionality, the FPGA serves as an SPI to PCIe converter for input data streams and vice versa for output data streams. Thus, a low-latency transformation from serial to parallel data as well as the other way round is implemented on the FPGA.

From the user's perspective, all these communication and conversion mechanisms are provided through PCIe API methods. Thus, low-level communication, FPGA-based data processing and external device interfacing are hidden from the control algorithm. This keeps the software simple, adaptable and thereby maintainable.

FPGA model	Xilinx Virtex 4 XC4VLX25-10FFG668C [4]
FPGA logic cells	~24.192
FPGA block RAM	864 KB
FPGA DSP blocks	48
Interfaces	PCIe x1, JTAG
PCIe Controller	PEX8311, PLX compatible [5]
GPIO Pins	64, 5V TTL compatible (via PIB64IO)

Table 2: PCIeV4Base FPGA board specifications

## 2.2 Physical layer – data acquisition, output and inter-module transport

For the developed system the ADS1281 high-resolution ADC from Texas Instruments is used as a first-stage data acquisition unit. The device's data sheet [6] promises a resolution of up to 31 bit with sample rates ranging up to 128 kSPS. Furthermore, the device comes with a programmable digital filter. Compared to other ADCs, the high resolution and sample rate of the ADS1281 make it possible to capture highly dynamic input signals. In our experiments, the ADS1281EVM evaluation board is used. It contains all support circuitry needed to operate the ADC. Additionally, it provides several voltage reference options as well as clocking options. [7]

As shown in figure 3 the ADC is directly connected to the GPIO pins of the FPGA. Therefore three master signals are provided by the FPGA application. These signals are distributed to all connected ADCs. MCLK is the master clock signal for the ADS1281. It has a frequency of 4 MHz, which is required for internal ADC operation. With the help of the MRST signal, all connected ADCs can be reset simultaneously. The operation of the ADCs can be synchronized with the help of the MSYNC signal, which is connected to the SYNC

input pin of every single ADS1281. The four remaining signals belong to the SPI [8] interface that is used to extract sampled data from the ADC. These signals are not shared among different ADCs. Every connected ADC needs a separate set of SPI signals connected to the FPGA. The FPGA acts as SPI master constraining the number of externally connectable components using the GPIO pins of the PCIEV4BASE FPGA board.

The use of a multiplexing approach could reduce the number of needed GPIO pins. In that case, the FPGA sequentially captures data from the connected ADCs by introducing a chip select signal that allows the SPI master to address different ADCs using the same signal lines.

However, this data acquisition method would introduce large latencies that are directly linked to the number of connected ADCs. Since low-latency data acquisition and processing are critical requirements for digital control systems, this method was discarded.

In contrast to using a multiplexing approach, the parallel interfacing method allows to simultaneously capture data from multiple connected ADCs. So the overall latency of the described data acquisition process is kept minimal and independent from the number of connected ADCs (see figure 4).

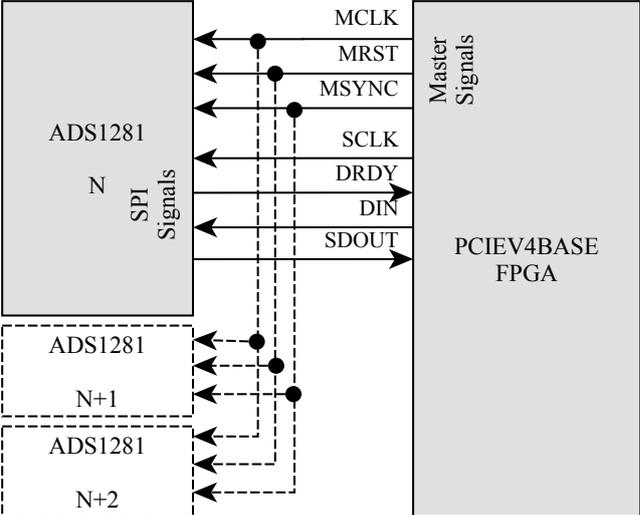


Figure 3: Block diagram of ADC-to-FPGA interface [6]

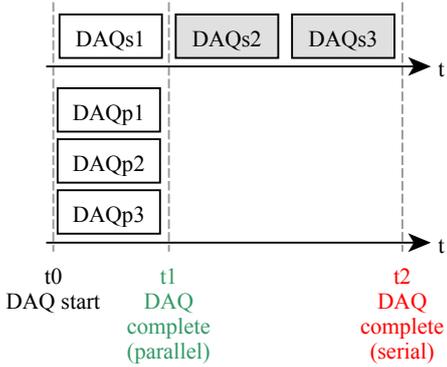


Figure 4: Parallel vs. serial DAQ

Control data output is organized in a similar way using AD5791 high precision, 20-bit DACs [9] on an EVAL-AD5791 board [10]. The serial interface between FPGA and DAC is illustrated in figure 5. Again, the interface consists of both master and SPI signals which are slightly different from the ADC-to-FPGA interface. As no master clock signal is needed to operate the DAC the MCLK signal is dispensable. All connected DACs can be reset synchronously by asserting the MRST signal; their respective output can be cleared using the MCLR signal.

In order to send digital output data to the AD5791 the following SPI signals are utilized: SCLK provides the clock signal with which data is clocked into the DAC circuit. During the development of the system a clock frequency of 5 MHz turned out to be a good tradeoff between reliability in terms of signal quality and transmission speed. Output data are serially transmitted to the DAC using the SDIN signal. With the help of the SYNC signal the boundaries of output data frames are indicated. For read back operations a SDOOUT signal is provided by the AD5791. In this application it is not necessary to read data back from the DAC, so we omitted this signal saving valuable GPIO pins of the FPGA for other purposes.

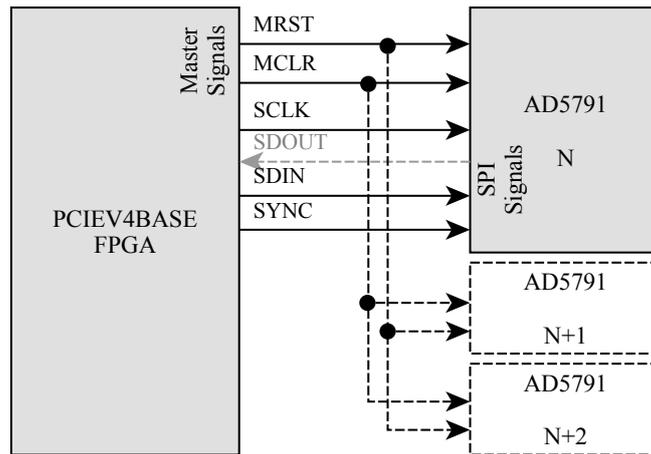


Figure 5: Block diagram of FPGA-to-DAC interface

Analogous to the data input mechanism, data output is organized to operate several DACs in parallel. This concept allows for a multi-channel data acquisition and output unit that provides constant latency despite an increasing channel count.

Necessary data conversion tasks, from serial to parallel data streams, are performed by the FPGA. These transformations are implemented in FPGA logic to be both time- and resource-efficient. Detailed information on the architecture of the FPGA application is presented in the following section.

For data transportation between FPGA and PC, PCIe is used. Compared to other common interfaces that can be found in standard PC systems, PCIe data transmissions come with lowest latency [11]. Because of that, PCIe is well-suited for the use in a PC based control system. In our case, the PEX8311 PCIe bridge chip is used, that is part of the PCIEV4BASE FPGA board. The PCIEV4BASE supports 32 bit single read/write and DMA single and continuous burst cycles for data transmission [2]. Preliminary tests showed that lowest latencies could be achieved using single read/write transmissions.

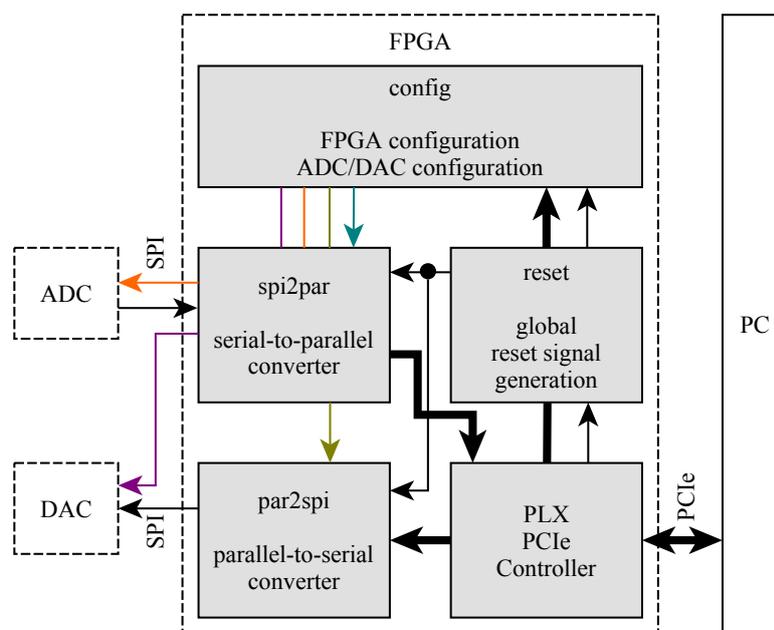


Figure 6: Top-level FPGA design block diagram

### 2.3 Management Layer – FPGA application architecture

Whereas data acquisition and data output belong to the physical layer of the proposed layer model of the demonstration system, the interface tasks executed on the FPGA are part of the management layer. Figure 6 shows the top-level design of the FPGA application giving an overview about the different FPGA modules and their interaction.

The presented FPGA design operates in a completely synchronous manner using the on board oscillators of the PCIEV4BASE FPGA board as clock signal sources. The PLX PCIe interface module uses the 66 MHz clock, which also serves as a clock source for the local bus interface between the FPGA and the PLX PCIe bus bridge PEX8311. Other clock signals are derived from the 200 MHz oscillator. Following the functionality of the different FPGA modules for configuration, reset signal generation, data conversion and PCIe interfacing is presented:

- **Configuration:** The “config” module of the FPGA design is responsible for the initialization and configuration of the FPGA conversion blocks as well as the connected ADCs. In order to allow for a very flexible configuration, all important configuration settings can be set from within the software application running on the PC. Thus, configuration data is transmitted from the PC to the FPGA via PCIe. In this process specially crafted configuration data packets consisting of a packet header and payload are used. They are received by the PLX PCIe controller and handed over to the FPGA’s configuration module for further processing (see figure 6).

Depending on the received packet header, the configuration module decides for which module and property the received setting is intended. Configuration options for the conversion modules are directly stored in the corresponding configuration registers (see green and aqua colored data streams in fig. 6). Settings for the ADC and DAC need to be serialized and transmitted to the connected device via SPI. The serial-to-parallel converter is used (see violet and orange data streams in fig. 6) for this task.

Moreover, the configuration module derives different clock signals from the 200 MHz clock source of the FPGA circuit and provides these signals for the other FPGA modules.

- **Reset generation:** The main reset signal for the FPGA system is the PCIe reset signal originating from the PLX module. It can be triggered from within the software application executed on the PC. The reset module on the FPGA generates separate reset and chip-enable signals for the configuration block and for the conversion modules. This assures that first of all the configuration takes place and the FPGA application is set into measurement mode afterwards. Once in this normal operation mode, no further configuration is possible due to the fact that transmitting configuration packets would interfere with the transmission of measurement data packets.
- **Serial-to-parallel conversion:** The serial-to-parallel conversion module “spi2par” is responsible for acquiring the ADC’s input data. It provides an SPI interface to external ADC devices as presented in section 2.2 of this paper (see fig. 3). A finite state machine (FSM) is used to read serial data from the SPI interface. Afterwards a 32 bit wide input data sample is compiled. This input sample is then placed into a First-In-First-Out storage block (FIFO) that can be accessed from the PCIe interface module. The FIFO element serves multiple purposes: It ensures data integrity of the input

samples when transferring data across different clock domains within the FPGA.<sup>1</sup> Furthermore control signals of the FIFO element are used in the PCIe interface module as data ready indicators.

- **Parallel-to-serial conversion:** Counterpart to the serial-to-parallel converter is the “par2spi” parallel-to-serial conversion module. It reads 32 bit wide data samples from another FIFO element and serially sends the data to the connected DAC using SPI (see fig. 5). Here, another FSM acts as controlling unit. The FIFO storage is filled with data received from the PCIe module. Thus, it acts as an interface component between the different clock domains of the PCIe module and the conversion block.
- **PCIe interface:** The “PLX” module provides the interface between FPGA and PC system. Data packets are read from the PCIe bus and written into the output data FIFO. Input data samples are read from the input FIFO provided through the serial-to-parallel converter and transmitted to the PC via the PCIe bus. As already mentioned, a data packet consists of 32 bits. A single read or write operation transmits a complete packet and is finished after a single clock cycle. Similar to the conversion modules the whole process is controlled by a FSM that evaluates FIFO and PCIe control signals and acts accordingly.

All described FPGA modules including their respective FSM controllers were implemented from scratch using the hardware description language VHDL. For the FIFO storage elements and the derivation of different clock signals intellectual property (IP) cores from Xilinx were utilized. These library components are specially designed for the target FPGA and thus provide both efficient and reliable function blocks. The usage of FSMs for control flow management, instead of using a soft-core processor implemented in FPGA hardware, leads to a resource efficient FPGA design as well as low-latency data processing.

## 2.4 Software API layer

Whereas the largest part, if not all, of the previously described physical and management layer is hidden from the control software application, the API layer provides all the needed functionality to the software developer. It was developed on top of both the CESYS’ Unified Development Kit (UDK) [2] and the PCI software development kit (SDK) provided by PLX Technology [12]. Both SDKs are available including source code files. Table 3 documents the exact SDK versions that were used. Since the CESYS UDK only supports Linux kernels up to version 2.6, minor modifications were necessary to build the PCIe kernel drivers for the used 3.8.9 kernel.

In order to provide easy to use access to the FPGA subsystem and data communications, several wrapper functions were introduced. The functions complete control system specific tasks by using UDK and PLX API calls themselves.

Development Kit	Version
CESYS UDK	2.1.0
PLX PCI SDK	6.4.0 (as part of CESYS UDK)

Table 3: Software development kit utilization

<sup>1</sup> Both converter blocks and the PCIe interface module are driven by clock signals with different frequencies. Special care must be taken when interchanging data between components of the different clock domains. FIFO elements with separately clocked read and write interfaces as used in the design allow for cross-domain data exchange.

UDK API calls are used to enumerate PCIe devices, to program and initialize the FPGA. Data transmissions are performed by calling low-level PLX API functions bypassing the UDK API. With this method PCIe communication latencies are kept at a minimum.

All of these PCIe API capabilities contribute to a very flexible and highly adaptable system as intended with the PC-based control system approach. Beyond that, it is very easy to access low-level devices from within the application layer. Control algorithm developers do not need to take care of internal data communication flows or hardware related issues. With that said, the presence of the FPGA application resident in the management layer is almost completely hidden from an application's point of view.

### 3. RESULTS AND DISCUSSION

System latency is one of the most critical properties in control applications since it is directly linked to achievable controller frequencies. In order to have the ability to control highly dynamic processes, high controller frequencies and thus low control system latencies are required. In the context of controlling EMFC balances, which is a desired application of the proposed system, controller frequencies in the order of several kilohertz are necessary. With the EMFC balance application in mind, a minimum controller frequency of 10 kHz was defined as a specific design goal.

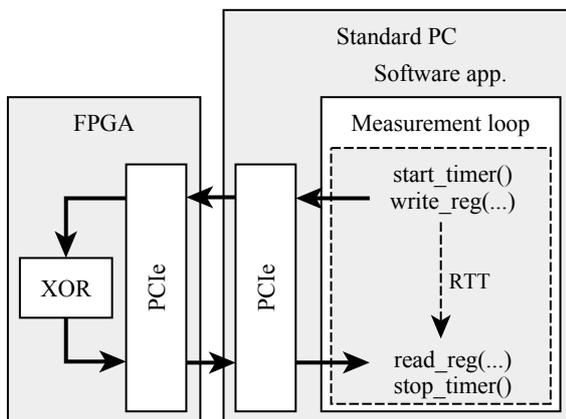


Figure 7: Measuring arrangement for determination of PCIe transmission RTT

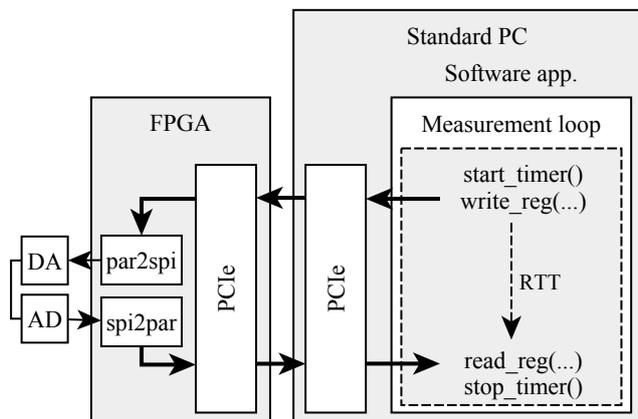


Figure 8: Measuring arrangement for closed loop latency determination

Accompanying to the entire design and development process, several latency measurements were conducted. The measurement illustrated in figure 7, gives an impression of the data communication performance between PC software and FPGA. During the measurement, data packets were sent from the PC system to the FPGA, which performed a simple exclusive-or operation and immediately returned the processed data packet to the PC. The integrity of the received data packets was checked within the PC software and the round trip time (RTT) of the whole process was measured. Using this arrangement, it was possible to characterize the system's latency (RTT), the error-rate of PCIe transmissions and the determinism of the controlling software executed on the PC system. The latter was possible by interpreting spikes in transmission latency.

Figure 9 shows the measurement results for a PC system according to table 1 that is running a standard Linux kernel. As one can see, the plot of the RTT versus the number of

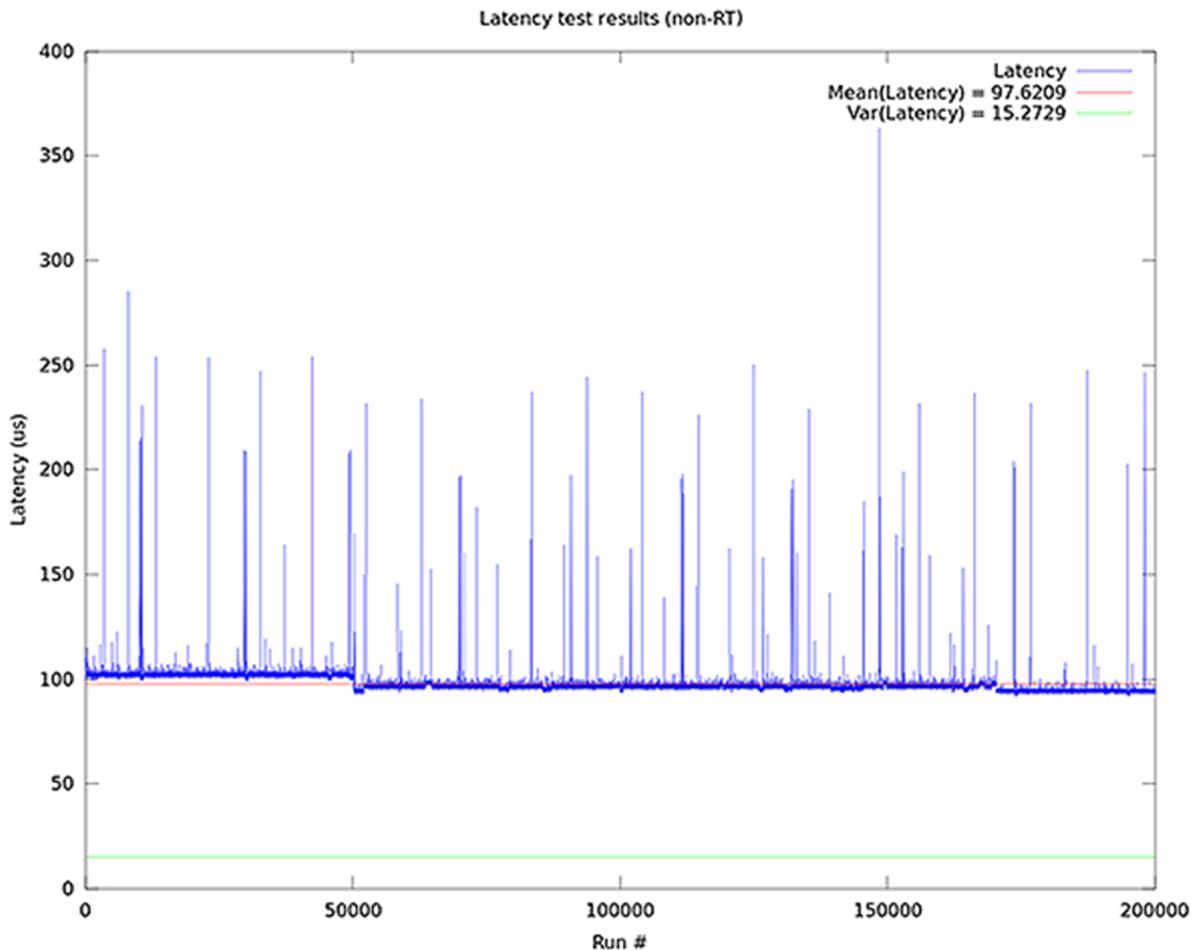


Figure 9: RTT measurement results for PC-FPGA transmissions (see fig. 7) on unpatched Linux system

measurement loop iterations is very spiky. Latency peaks of up to 400 microseconds occurred. With these peaks, the design goal could not be met and further optimizations were necessary.

That is why the measurement was repeated after the RT Linux patch was applied to the operating system. Furthermore the software was optimized to solely use low-level PCIe functions to avoid unnecessary overhead that might be introduced by using higher-level APIs. As shown in figure 10, these adjustments lead to large improvements: the RTT times decreased as well as the abundance of peaks and their duration. Worst-case RTT values of PC-to-FPGA transmissions remained below 4.5 microseconds.

After these initial tests, the ADC and DAC devices as described in section 2 were connected to the system. Figure 8 illustrates this test setup that allows closed loop latency measurements. Here, voltage values are generated in the PC software and are transmitted to the FPGA as output values for the DAC. The analog voltage levels are continuously sampled by the ADC, whose analog inputs are directly connected to the analog outputs of the DAC. With the help of the FPGA application, input data is converted into a PCIe data stream that is read from within the software. Again, the latency was measured by sending single data packets through the system and measuring their specific RTT.

Initially, the ADC was sampling input data at a rate of 32000 samples per second (SPS). In this case all 31 data bits of the input data samples, provided by the ADC, were transmitted to the software application. The latency measured for a single data packet travelling through the whole system was around 142 microseconds (see table 4). Due to the fact that these latency values would not allow for closed loop frequencies of 10 kHz, the causes for these high RTT values were investigated.

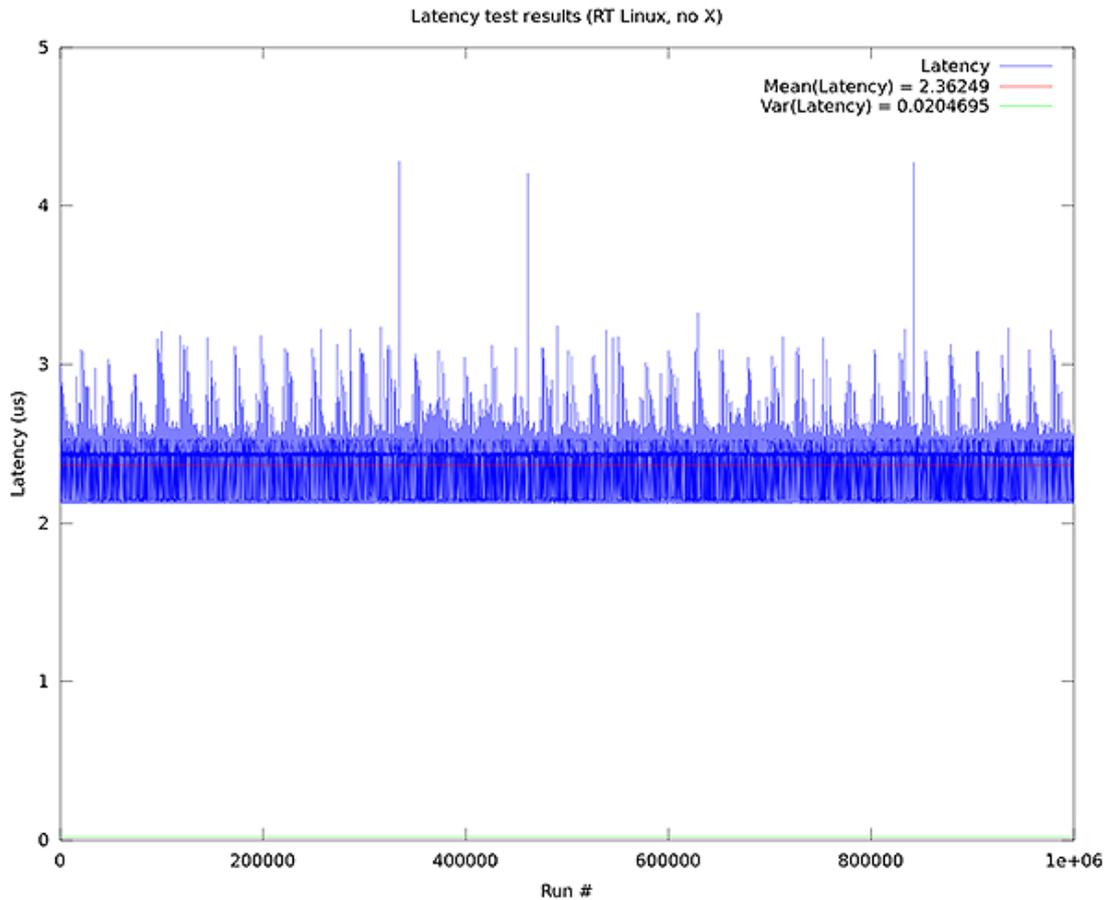


Figure 10: RTT measurement results for PC-FPGA transmissions (see fig. 7) on RT Linux system

In order to find the root cause of these closed loop latencies, the sample time of the ADC and the time needed to output an analog voltage value through the DAC were identified. With a latency of 7 microseconds, of which the largest part is induced by the SPI data transmission, the DAC could be excluded as being the component introducing high latency into the system.

Further tests with the ADS1281 ADC showed that it took around 120 microseconds in the initial configuration to sample a particular analog voltage level and read it via SPI. To reduce ADC latencies, the sample rate was doubled to 64 kSPS. Furthermore the FPGA application was configured to read only the 24 most significant bits (MSB) of sampled data to reduce the time needed for the SPI transmission. These adaptations lead to a significant latency decrease of the closed loop. As summarized in table 4 the closed loop RTT values could be reduced to around 80 microseconds meeting the initial control frequency design goal.

At the time of publishing this paper, the system is yet being actively developed and optimized. Thus, latency measurements and considerations are still part of current research.

ADC sample rate (kSPS)	ADC data bits	RTT ( $\mu$ s)
32	31	142
64	24	80

Table 4: Closed loop latency measurement results (see fig. 8)

## 4. CONCLUSIONS

In this paper we presented a processing platform for digital controllers based on a standard PC system. Data acquisition and data output were realized using a PCIe-enabled FPGA board allowing for low-latency data transfers between connected ADC as well as DAC devices and the PC. Both control algorithm development and execution are performed in software to guarantee a high degree of flexibility and adaptability. A special software API was developed that enables control algorithm developers to efficiently initialize, configure and use the proposed system without the need to care about detailed low-level characteristics of used hardware components.

The system uses a state-of-the-art Linux operating system that was enhanced using RT Linux kernel patches in order to provide real-time capabilities for control algorithm execution. The impact of these modifications on control system latency was analyzed.

In order to utilize the proposed system for controlling of highly dynamic processes, like EMFC balances, a controller frequency design goal of 10 kHz was set. This requirement was met by optimizing both the developed software API and the FPGA application. Specifically, closed loop frequencies of about 12.5 kHz were achieved for a single channel system.

## ACKNOWLEDGEMENTS



This research was done within the framework of the InnoProfile-Transfer project (see: [www.tu-ilmenau.de/ikwi](http://www.tu-ilmenau.de/ikwi)). The authors gratefully acknowledge the financial support for this project by the German Federal Ministry of Education and Research (BMBF) in cooperation with Sartorius Lab Instruments GmbH & Co. KG, SIOS Meßtechnik GmbH, PAARI Waagen- und Anlagenbau GmbH and driveXpert GmbH.

## REFERENCES

- [1] H. Zimmermann, “OSI Reference Model – The ISO Model of Architecture for Open Systems Interconnection”, IEEE Transactions on Communications, vol. COM-28, no. 4, pp. 425-432, April 1980.
- [2] CESYS Gesellschaft für angewandte Mikroelektronik mbH, “PCIEV4BASE – Virtex-4 FPGA board with PCIe interface“, Herzogenaurach, Germany, June 2010.
- [3] CESYS Gesellschaft für angewandte Mikroelektronik mbH, “PIB64IO – Plugable Interface board with 64 digital channels“, Herzogenaurach, Germany, November 2007.
- [4] Xilinx Inc., “Virtex-4 FPGA Data Sheet: DC and Switching Characteristics“, San Jose, U.S.A., September 2009.
- [5] PLX Technology Inc., „ExpressLane PEX 8311AA PCI Express-to-Generic Local Bus Bridge Data Book“, Sunnyvale, U.S.A., December 2009.
- [6] Texas Instruments Inc., “ADS1281 – High-Resolution Analog-to-Digital Converter“, Dallas, U.S.A., June 2010.
- [7] Texas Instruments Inc., “ADS1281EVM and ADS1281EVM-PDK User’s Guide“, Dallas, U.S.A., May 2011.
- [8] Motorola Inc., “SPI Block Guide V03.06“, Chicago U.S.A, February 2003.
- [9] Analog Devices, “AD5791: 1ppm 20-Bit Voltage Output DAC Datasheet (Rev. D)“, Norwood, U.S.A., July 2013.

- [10] Analog Devices, “Evaluation Board for a 20-Bit, Serial Input, Voltage Output DAC – Evaluation Board User Guide UG-185 (Rev. 0)”, Norwood, U.S.A., 2010.
- [11] S. Basu, “Architecting High-Speed Data Streaming Systems”, National Instruments Technical Symposium, May 2009.
- [12] PLX Technology, „PCI SDK Software Development Kit – Reference Manual“, Sunnyvale, U.S.A., May 2003.

## CONTACTS

M.Sc. F. Schwesinger  
Dipl.-Ing. G. Krapf  
Univ.-Prof. Dr.-Ing. habil. T. Fröhlich

[folker.schwesinger@tu-ilmenau.de](mailto:folker.schwesinger@tu-ilmenau.de)  
[gunter.krapf@tu-ilmenau.de](mailto:gunter.krapf@tu-ilmenau.de)  
[thomas.froehlich@tu-ilmenau.de](mailto:thomas.froehlich@tu-ilmenau.de)