Use of hardware and software information processing in mechatronics on the example of an adaptive control

Thomas Reinemann
Otto-von-Guericke University Magdeburg
Insitute of Mechatronics and Drive Technology
Universitätsplatz 2 , D-39106 Magdeburg, Germany
thomas.reinemann@mb.uni-magdeburg.de

Roland Kasper
Otto-von-Guericke University Magdeburg
Insitute of Mechatronics and Drive Technology
Universitätsplatz 2 , D-39106 Magdeburg, Germany
roland.kasper@mb.uni-magdeburg.de

***Abstract*–Today micro-controllers and signal processors are the standard-implementing platform for embedded controllers. But often their performance can not keep pace with requirements for demanding applications, for example in mechatronic systems. Usually, in this case the micro-controller is replaced by a faster one. But this doesn't avoid software problems arising in realtime applications even by using high level languages and always by using assembler language. Further problems are concerned with higher clock frequencies and energy consumption.**
On the other hand FPGAs are very well suited for high speed applications, because information processing can be performed in parallel. But they usually lack common mathematical function libraries, so that control and filter algorithms have to be implemented in a time consuming and expensive process.
Today's FPGAs offer a huge number of gates, so that even a processor core can be implemented inside. This enables a combination of hardware and software signal processing within only one IC. Thereby the available software can be used further on and only some algorithms requiring high performance have to be implemented in hardware. Furthermore some aspects of reconfigurable computing are treated.

1 Introduction

The rapid development of microelectronics in general and of programmable logic devices (FPGAs and CPLDs) in special, opens completely new possibilities of digital implementation of control and signal processing in embedded systems. The success of microcontrollers and digital signal processors was characterized by an efficient hardware combined with flexible and adaptable software structures. The strength of programmable logic devices is given by flexible and very fast hardware structures. Together these technologies show the way in a new era of reconfigurable controllers. These allow the adaptation of hardware-based controllers to specific operating conditions by changing the hardware configuration during operation time by which the necessary number of gates and the power demand is reduced. A good overview of the potentials and implementation aspects of reconfigurable computing is given in [1], [2].
Some dedicated considerations emphasize the parallelizing at block level and serializing at signal level in [2], which lead to very efficient solutions.

The development of universal, usable dynamically reconfigurable computer architectures introduces a number of difficult questions. The dynamic reconfiguration of the system is enabled by exchange of some components during run time. Besides, reconfiguration often requires some kind of operating system which controls and supervises the reconfiguration process.

The base for the reconfiguration process is build by FPGA configuration files, which represent the resources needed by an application. They are used for a dynamic allocation of required resources. Indeed, the reconfiguration ability also causes costs. Generally, for the solution of a problem several tasks are necessary. Therefore it is to be ensured that all these tasks can be loaded and linked at the same time.

However hardware implementation isn't a general solution at all. Since sequential algorithms can be better implemented in software. Therefore, a combination of processor and programmable logic offers better performance. To increase the flexibility, it is smarter to integrate a processor core into the PLD as a soft core, than to add an additional external processor. The duration of the product cycles in the mechanical engineering and automotive industry is significant higher than those in the IT industry. As a consequence, processors and other integrated circuits used for the first development are no longer competitive or even no more in trade later in product life cycle.

Presently the specification of control algorithms for hardware is made on the base of hardware description languages (HDL). Nevertheless, this is not the method for the development engineer in the field of mechatronics, who prefers tools like block diagram editors etc. Also HDLs provide arithmetic functions only on a low level. These are to provide by the developer or PLD manufacturer themselves and therefore there is no uniform library of arithmetic functions which supports portability. Arithmetic elements provided, generally use bit-parallel arithmetic in conjunction with logic-intensive and complex operators. Therefore a method has been developed in [3] for the specification and synthesis of control algorithms in a bit-serial manner. A block diagram editor is used there for specification [4], which is build upon a library of bit-serial arithmetic operators [5].

2 Reconfiguration

Reconfiguration means changing logic parts of an FPGA. This can happen during run time. This case is usually called dynamic reconfiguration. Off line it is referred to as static reconfiguration. During reconfiguration generally some parts remain unchanged. This is on the one hand the logic device with its supporting devices and on the other hand some parts of the implemented user logic.
Changes which result from modified requirements, improved functionality or the elimination of errors can be allowed by static configuration by re-use of existing hardware. This is especially interesting against the background of reconfigurable analogous and digital interfaces which can be reconfigured to work with different clock cycles, word lengths and protocols. Thereby the possibility exists to carry out the ability of configuration off the PLD and to develop an universal hardware (PCB), which can host several generations of algorithms alternatively. By this, the total number of hardware variants is drastically reduced. This is important e.g. in the life cycle of a product of the mechanical or automotive industry, which lasts from the start of development, over production and until the end of the supply of spare parts up to 30 years. In contrast, the product cycles of the IT industry are much shorter. Against this background, the supply with programmable devices (CPUs, periphery ICs) represents a big problem. Storing these devices over such a long period is very expensive, particularly as storage failures are to be deplored. A universal configurable hardware reduces this problem clearly.

Generally, different algorithms are processed using controllers depending on the operating point or operating state. An example is a vehicle's ABS, which has to run very complicated control functions during the braking process but in the remaining time its main task is supervision. Often functions are related to external events, what makes the performance requirement strongly depending on the respective operating state. As an example an injection system states here, which has to run certain functions dependent on the engine speed at defined piston positions. This leads among other things to the fact that at high motor speed almost the complete performance of the processor is used for this task only.

Presently there is still no hardware platform which permits the dynamic reconfiguration simply and in real time. In principle the reconfiguration time increases with the amount of the reconfiguration, because according to this more information has to be transferred. Some FPGA types already support a partial reconfiguration. However, this cannot occur arbitrarily, but only by lines or columns of the underlying array. Another problem represents the signal consistency, in particular if values from several sampling times have to be processed, or the calculation takes a longer time than one sampling period. Presently an algorithm runs on, even if its results are not actually used. Hence, at the switching point the results of the algorithms to be switched, are close together and unsteadiness is avoided therefore. If now the algorithms are exchanged, no calculation occurs in the rests. This brings on the problem of the unsteadiness avoidance in the switching point.

3 Example

The application possibilities of modern FPGAs should be shown at the example of a track-controlled vehicle model. The vehicle follows a mark on the ground. This can be a colour mark as well as a magnetic track. According to the type of the mark, an appropriate sensor has to be connected together with a suitable evaluation of the signals. To make the vehicle follow a marked path, all deviations of the vehicle to the track should be compensated. For example, a CCD line [6] can be used for an optical recognition of the control deviation e (see Fig. 1). It has a large number of photosensitive photodiodes (e.g., 256) which leads to an accordingly high sampling rate (e.g., 256 x 1 kHz). However, a low resolution can be sufficient in kind of a simple bright-dark differentiation (1 bit resolution). As another approach, two magnetic field sensors can be used to sense a magnetic track. These offer a lower sampling rate (e.g. 2 x 1 kHz), but need a much higher resolution (e.g., 12 bits), because the signal amplitude corresponds to the distance of the sensor and the track.

4 The model

The following simple model of the steering system represents the basis for the implementation (see Fig. 1). If Δx is the path covered within a time unit ΔT (proportionally to the vehicle speed v) and α is the steering angle, the movement of the wheel (Δy) is given by

$$\begin{aligned} \Delta y &= f(\Delta x, \alpha) \\ \Delta y(k) &= \Delta x(k) \cdot \tan\alpha(k) \end{aligned} \tag{1}$$

This approximation is only valid for sufficiently small time intervals ΔT, then a steady speed can be assumed during one time interval. For usual applications this condition is fulfilled. Because Δy depends on the steering angle as well as on the covered path we see a non-linear (variable gain) behaviour of the controlled system.

For Δx as well as for Δy is assumed that one digit corresponds to path of one millimetre. Such scaling definitions are important in particular against the background of reconfiguration. Since scaling influences the controller's dimensioning and permits only an easy

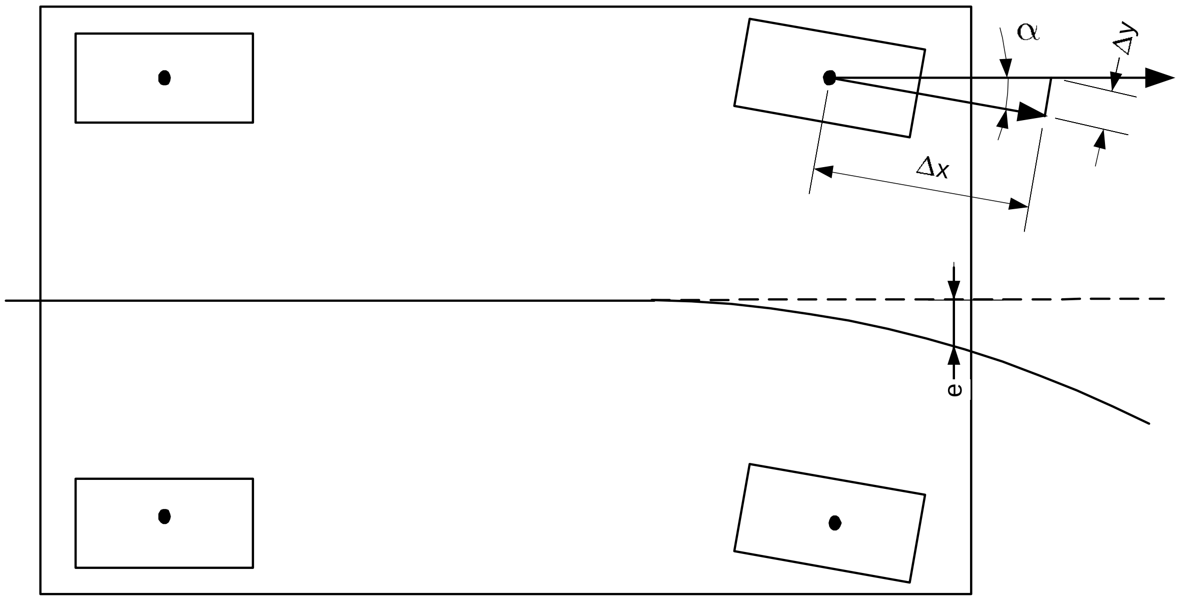

Fig. 1: Definition of the items

exchange of the sensors and actuators if the components to be exchanged have the same gain.
Due to the non-linearity of the controlled system, the speed affects the manipulated variable, via the controller's parameters. Consequently these parameters are adopted by the car's speed.
For the adaptation the poles of the whole system are placed to suitable positions and the controller parameters are derived from that.

Assuming that u(k)=tan $\alpha(k)$ and Δx=const one can form a transfer function from eq. (1)

$$G_S(z) = \frac{\Delta y}{u} = \Delta x \qquad (2)$$

for the controlled system G_S. The transfer function G_C for a discrete PID controller is given by eq. (3):

$$G_C(z) = \frac{u}{e} = K_P + \frac{z-1}{z} K_D + \frac{z}{z-1} K_I \qquad (3)$$

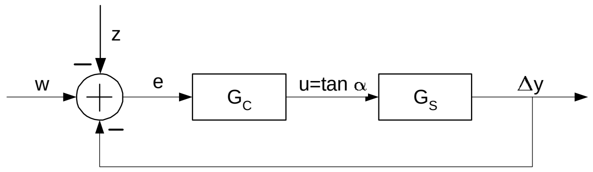

Fig. 2: Control loop

Fig. 2 shows the control loop and accordingly to this figure a closed loop transfer function G can be established using eq. (2) and (3). This transfer function has the following denominator G_D:

$$G_D = [(K_P + K_D + K_I)\Delta x + 1]z^2 - [(K_P + 2K_D)\Delta x + 1]z + K_D \Delta x \qquad (4)$$

On the other hand, a second order system with 2 poles z_1 has the following transfer function denominator G_{2OD}:

$$G_{2OD} = (z - z_1)^2 = z^2 - 2z_1 z + z_1^2 \qquad (5)$$

Eq. (4) and (5) offer the possibility to compare the coefficients to find values of the controller parameters depending on the placement of the pole z_1.

$$K_D = \frac{z_1^2}{\Delta x} \qquad (6)$$

$$K_P = \frac{-2z_1^2 + 2z_1 - 1}{\Delta x} \qquad (7)$$

$$K_I = -K_D - K_P = (z_1 - 1)^2 \qquad (8)$$

These three equations come into operation to adopt the controller's gains depending on the car's speed.

5 Simulation

The test bench shown in Fig. 3 is used to embed the system, the controller (Fig. 5) and the adaptation (Fig. 4) for simulation via Matlab/Simulink. In a first step a PID-Controller came into operation. However, the values of the D-part was very small compared to the P and I values. Therefore the D-part was taken away. The simulation results are given in Fig. 6 and Fig. 7. Fig. 6 presents the control deviation depending on the disturbance z. An increasing disturbance represents a tighter curve and can't be fully compensated, this has its reason by the simple controller structure.

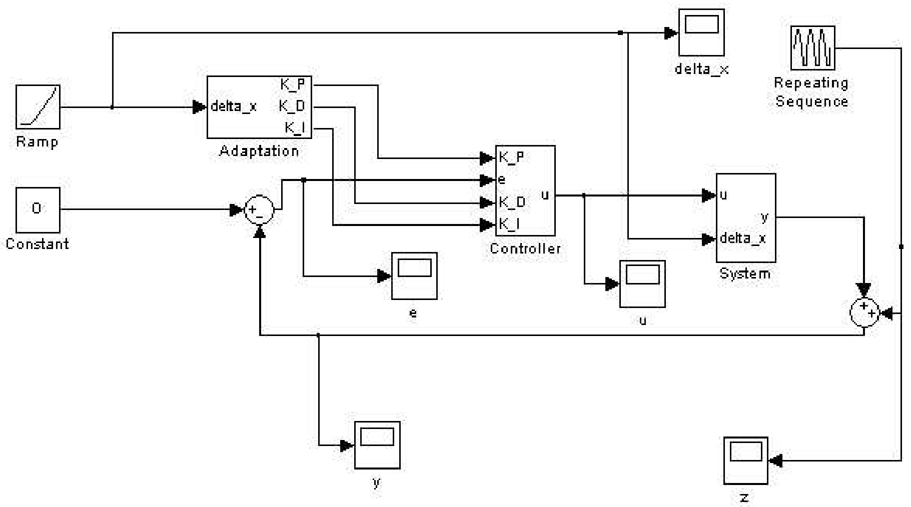

Fig. 3: Testbench

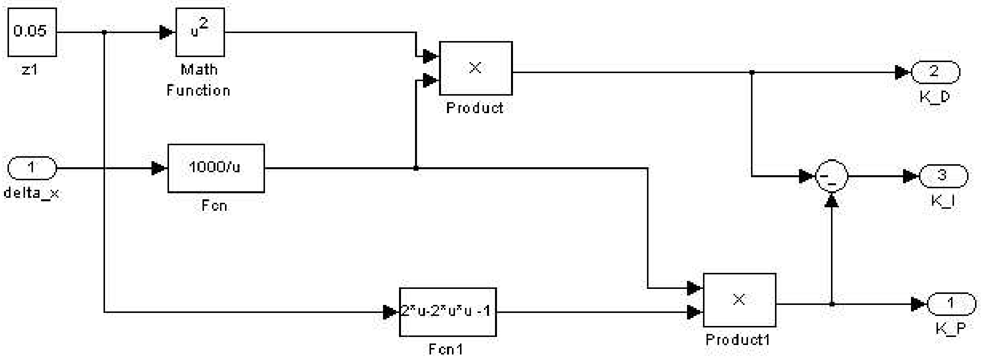

Fig. 4: Adaptation

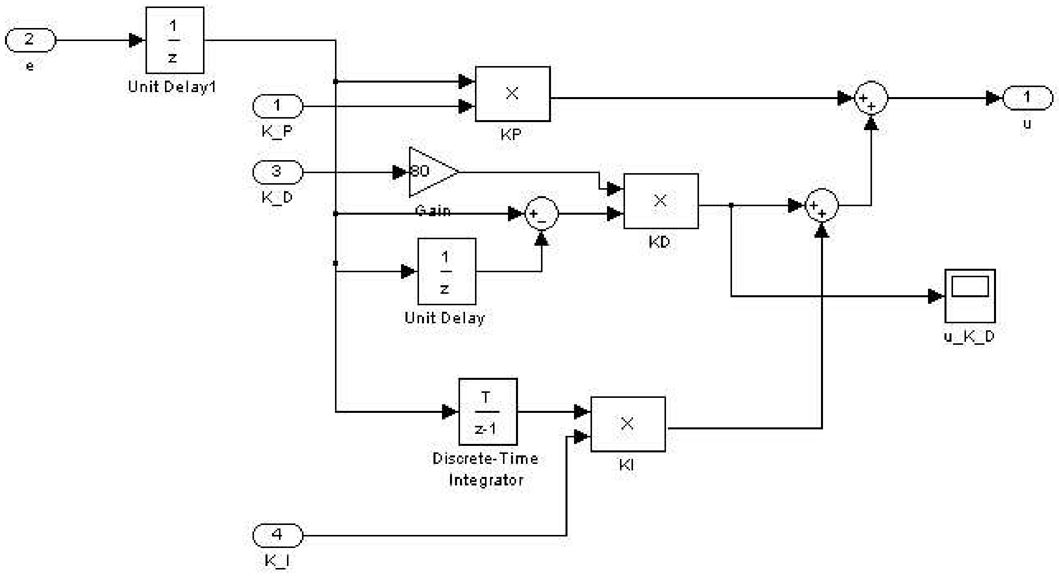

Fig. 5: PID-Controller

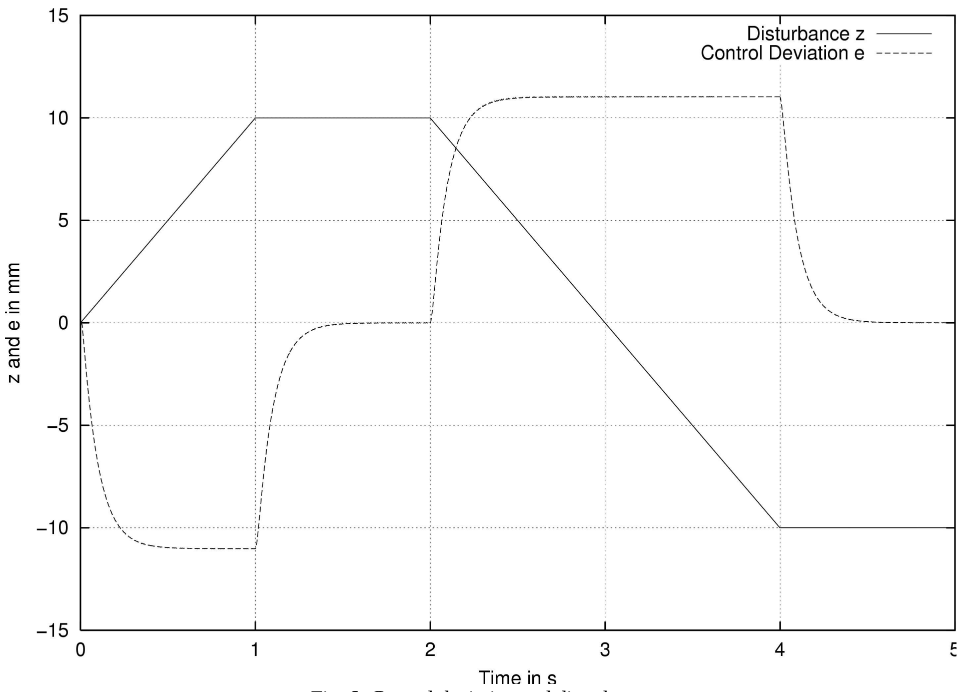

Fig. 6: Control deviation and disturbance

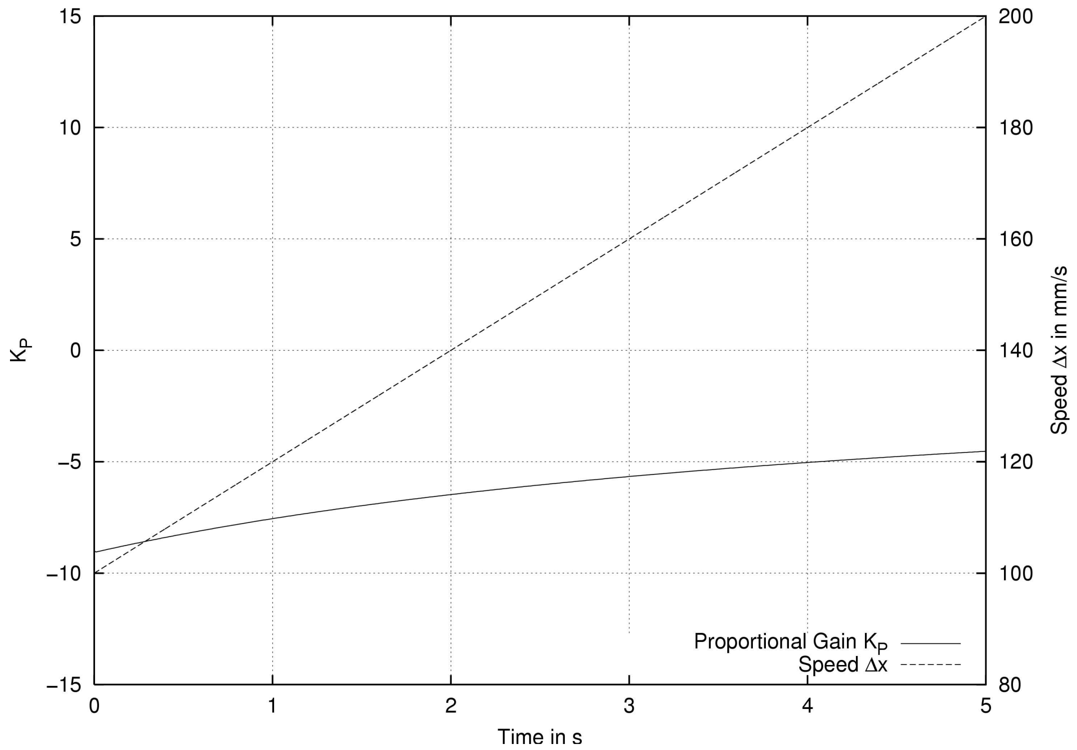

Fig. 7: Car speed in connection with controller gain K_P

However a constant disturbance is compensated by the I-part of the controller.
The controller gain decreases with the increasing car's speed to balance the steer gain, this shows Fig. 7.

6 Implementation

The controller and adaptation is implemented in an FPGA using VHDL. However, to hide the HDL for specification, a block diagram editor (HDL Designer by Modeltech) [4] came into operation. For this editor a library of signal processing components has been developed by the authors in prior work [5]. These components use bit serial information processing and transmission. This means the operands are transmitted and processed in time division. The LSB is transmitted as the first bit, because carry most propagates from LSB to MSB in arithmetic operations. Bit serial signal processing reduces the complexity and logic effort

significantly at a similar performance compared to bit parallel signal processing.
However non linear operations, as a division, are difficult to implement, usually a look-up-table is used. To avoid this and to gain experience of processors implemented within an FPGA, a processor core is used for implementation of the adaptation process. For this example the free i8051 core from the Dalton project [5] comes into operation. This core is equal to the original i8051 from Intel. The processor reads Δx, converts it to a floating point number and computes the gains. Since the gains have a limited range ($0 < K_I < 10$ and $-10 < K_P < 0$), see simulation, eight bit integers are used for their representation. They are fed to the controller via the processor's ports.
The controller uses these gains for the computation of an appropriate value of the controlled variable.
The sampling period of the slower changing car's speed is 10 ms, compared to 1 ms for the control deviation.
The processor core needs about 1400 FlipFlops, however the majority of these FFs is used to store the program code. The necessary PID controller utilizes only about 200 FFs. This limited need for logic enables the circuit to fit in a small XILINX-FPGA [8].

7 Conclusion

The paper shows the possibilities offered by modern FPGA hardware and development tools for the implementation of fast and complex control algorithms. The high amount of logic of today's FPGAs allows the integration of processor cores to run slow but complex algorithms in software together with high speed hardware algorithms. This new degree of freedom in balancing the parts that are implemented in hardware or software respectively, together with the possibility to reconfigure hardware structures as easily as software, will lead to a complete new generation of control systems. The example presented in this paper, only gives a first impression of the ideas that can be realized now.

References

[1] Paul Master: The Age of Adaptive Computing Is Here, LNCS 2438, p. 1 ff.

[2] Andre DeHon, John Wawrzynek: Reconfigurable computing: what, why, and implications for design automation, Proceedings of the 36th ACM/IEEE conference on Design automation conference, 1999, ISBN 1-58133-109-7, p. 610 – 615, New Orleans, Louisiana, United States, ACM Press.

[3] Reinemann, Th.; Kasper, R.: High Speed Implementation of Controllers and Filters for Mechatronic Systems; http://www.techonline.com/community/home/14817, TechOnline

[4] MentorGaphics: HDL Designer Series, http://www.mentor.com/hdldesigner/index.cfm?mode=fpga

[5] Kasper, Roland; Reinemann, Thomas: Gate level implementation of high speed controllers and filters for mechatronic systems Mechatronic Workshop 2000; Krakau;

[6] Foto Digital Service: http://www.foto-digital.de/glossar_c.html

[7] University of California Dept. of Computer Science Dalton Project http://www.cs.ucr.edu/~dalton/8051/

[8] Auer, A; Rudolf, D. J.: FPGA Feldprogrammierbare Gate Arrays; Huethig; Heidelberg; 1995