EURASIP Journal on Applied Signal Processing 2003:6, 594–602

Design and Implementation of Digital Linear Control Systems on Reconfigurable Hardware

Marcus Bednara
Department of Computer Sciences 12, Hardware-Software-Co-Design, Friedrich-Alexander-Universität Erlangen, D-91058 Erlangen, Germany
Email: bednara@informatik.uni-erlangen.de

Klaus Danne
Heinz Nixdorf Institute, University of Paderborn, D-33102 Paderborn, Germany
Email: danne@upb.de

Markus Deppe
Mechatronic Laboratory Paderborn (MLaP), University of Paderborn, D-33098 Paderborn, Germany
Email: Markus.Deppe@MLaP.de

Oliver Oberschelp
Mechatronic Laboratory Paderborn (MLaP), University of Paderborn, D-33098 Paderborn, Germany
Email: Oliver.Oberschelp@MLaP.de

Frank Slomka
Department of Computer Science, Embedded Hardware/Software Systems Group, Carl von Ossietzky Universität Oldenburg, D-26111 Oldenburg, Germany
Email: frank.slomka@informatik.uni-oldenburg.de

Jürgen Teich
Department of Computer Sciences 12, Hardware-Software-Co-Design, Friedrich-Alexander-Universität Erlangen, D-91058 Erlangen, Germany
Email: teich@informatik.uni-erlangen.de

Received 14 March 2002 and in revised form 15 October 2002

The implementation of large linear control systems requires a high amount of digital signal processing. Here, we show that reconfigurable hardware allows the design of fast yet flexible control systems. After discussing the basic concepts for the design and implementation of digital controllers for mechatronic systems, a new general and automated design flow starting from a system of differential equations to application-specific hardware implementation is presented. The advances of reconfigurable hardware as a target technology for linear controllers is discussed. In a case study, we compare the new hardware approach for implementing linear controllers with a software implementation.

Keywords and phrases: digital linear control, reconfigurable hardware, mechatronic systems.

1. INTRODUCTION

Modern controller design methods try to support the design of controllers at least semiautomatically. The need for a transparent and straightforward design process often leads to software implementations of controllers, that is, microprocessor programs specified in a high-level language using floating-point arithmetic. This approach, however, is inappropriate for applications with high sampling rates ($f_s > 20\,\text{kHz}$). Such applications are typically micromechanic systems like hard disk drives [1, 2, 3]. Exploding density of the hard disks requires controllers with enhanced accuracy. This leads to very high sampling rates. Here, FPGA technology is a way to perform high-speed controllers with high flexibility. With high-level design tools such as VHDL and logic-synthesis CAD tools and FPGA as target technology, a rapid prototyping of complex linear control systems becomes possible. Low-cost FPGA will allow their use in the final product in the near future. To support the use of hardware implementations, however, new automated design flow methods are required.

The advances in silicon technology and the high computational power of modern microprocessors and DSPs allow for implementation of flexible linear controllers in software. However, the implementation of state-space controllers for applications with high sample rates requires short computational times. As the number of required calculations grows nonlinearly with the number of states, application-specific hardware is often unavoidable to provide sufficient computational power. Yet dedicated hardware is very inflexible since it is impossible to adapt the implementation on changing requirements, new applications, or modified parameters. Reconfigurable hardware structures provide a way out of this dilemma. With reconfigurable hardware, it is possible to design an application-specific hardware along with the high flexibility of software solutions. For linear controllers, parallelism can be used as needed and the implementation can be changed if required.

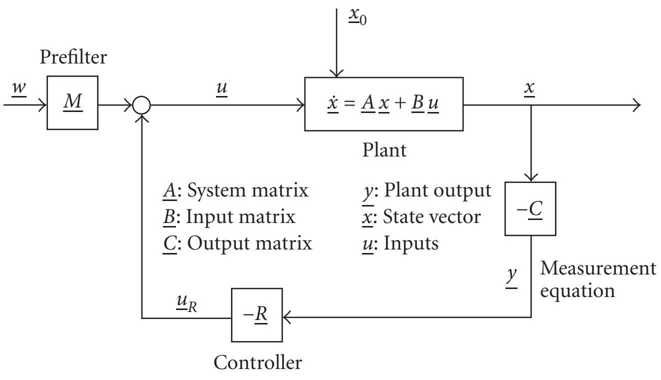
In this paper, we describe an approach for an automated mapping of linear controllers to reconfigurable hardware. Furthermore, we quantitatively compare such solutions to software implementations. We develop a generic hardware structure which can be easily adapted to new applications. In difference to [4], where a special instruction set processor for implementing digital control algorithms is described, our approach implements all parts of the controller in hardware.

Important issues for using reconfigurable hardware are:

(1) What speedup can be obtained by the use of hardware as compared to a pure software solution?
(2) Do typical control systems fit current FPGA devices?

As a case study, we have implemented a linear controller for an inverse pendulum in hardware and software on an FPGA-based reconfigurable hardware platform and have compared the results. The experiments show the potential of reconfigurable hardware to implement fast and flexible solutions of linear control systems. Compared to pure software solutions which can also change the controller parameters during runtime, the new approach [5] has several advantages.

(1) The obtainable sample period only scales linearly with the problem size which allows for controller implementations with very high sample rates.
(2) FPGAs offer the same flexibility as software implementations along with the speed of application-specific hardware.
(3) If the applications require higher clock rates as supported by the used FPGA technology, it is very easy to adapt the designed hardware to other faster silicon technologies such as gate arrays.
(4) By implementing different controllers in parallel for the same application, it could become very easy to switch between the controllers to adapt the system to changing-environmental parameters. By proper blending mechanisms, the controller will not remain in an undefined state during switching.



$\underline{x}_0$
Prefilter
$\underline{w}$
$\underline{M}$
$\underline{u}$
$\dot{\underline{x}} = \underline{A}\,\underline{x} + \underline{B}\,\underline{u}$
$\underline{x}$
Plant
$\underline{A}$: System matrix
$\underline{B}$: Input matrix
$\underline{C}$: Output matrix
$\underline{y}$: Plant output
$\underline{x}$: State vector
$\underline{u}$: Inputs
$-\underline{C}$
$\underline{y}$
Measurement equation
$\underline{u}_R$
$-\underline{R}$
Controller

Figure 1: General structure of a control system.

Especially the last item will be the subject of our future work. The paper is organized as follows. In Section 2, we give a basic overview of the mathematical principles of digital linear control systems design. The design flow for the implementation of linear systems of differential equations in reconfigurable hardware is described in Section 3. A description of the proposed architecture of the software and hardware implementation is given in Section 4. Section 5 introduces a case study on how linear controllers can be implemented on FPGAs and describes the complete design flow for the example. In this section, we also compare a software implementation of the example with the pure hardware solution. We conclude with a discussion of future work in Section 6.

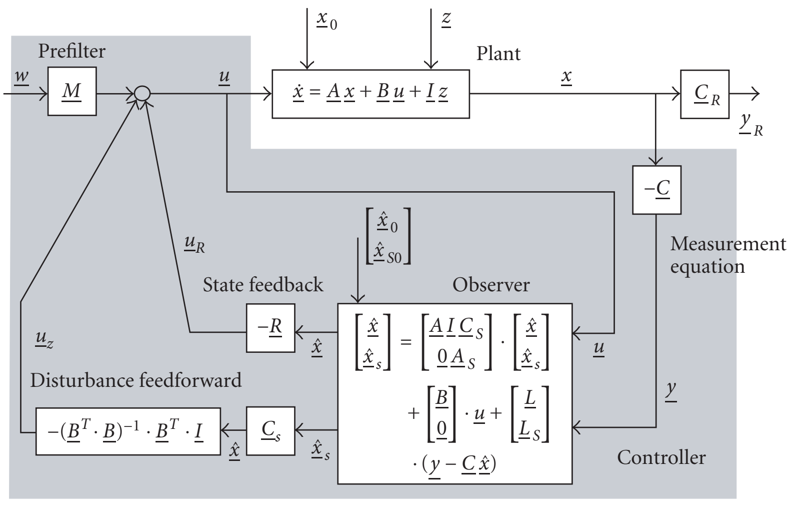
2. LINEAR CONTROLLERS

2.1. Structure

The basic idea of controlling a system (called control path or plant) is to take influence on its dynamic behavior via a control feedback loop. A controller takes measurements from the control path and computes new input variables to the system. This results in a typical feedback structure is shown in Figure 1. Generally, the system consisting of controller and control path is continuous, nonlinear, and time variant. In most cases, however, the controller and control path can be modeled as linear time-invariant systems (see Figure 1), where the plant is specified by a system of linear differential equations.

2.2. Mathematical foundations

In order to explain our methodology, we start from the general controller structure in Figure 2, which shows a multivariable feedback controller with plant [6]. The essential parts of the multivariable controller are the state feedback, the disturbance rejection, and the observer. An observer is used to reconstruct states that could not be measured and it has the same order as the plant itself. The observer consists of a model of the plant and a model of the disturbance which


Prefilter
Plant
State feedback
Observer
Measurement equation
Disturbance feedforward
Controller
$\dot{\underline{x}} = \underline{A}\,\underline{x} + \underline{B}\,\underline{u} + \underline{I}\,\underline{z}$
$-(\underline{B}^T \cdot \underline{B})^{-1} \cdot \underline{B}^T \cdot \underline{I}$
$\begin{bmatrix}\dot{\hat{\underline{x}}} \\ \dot{\hat{\underline{x}}}_s\end{bmatrix} = \begin{bmatrix}\underline{A}\,\underline{I}\,\underline{C}_S \\ 0\,\underline{A}_S\end{bmatrix} \cdot \begin{bmatrix}\hat{\underline{x}} \\ \hat{\underline{x}}_s\end{bmatrix} + \begin{bmatrix}\underline{B} \\ 0\end{bmatrix} \cdot \underline{u} + \begin{bmatrix}\underline{L} \\ \underline{L}_S\end{bmatrix} \cdot (\underline{y} - \underline{C}\,\hat{\underline{x}})$

$\underline{A}$: System matrix	$\underline{M}$: Prefilter matrix	$\underline{x}_0$: States of plant
$\underline{B}$: Input matrix	$\underline{R}$: Recoefficient matrix	$\hat{\underline{x}}$: States of observer
$\underline{C}$: Output matrix	$\underline{u}$: Inputs	$\hat{\underline{x}}_s$: States of disturbance observer
$\underline{I}$: Identity matrix	$\underline{w}$: Command variable	$\underline{y}$: Plant outputs
$\underline{L}$: Observer matrix	$\underline{x}$: States of plant	$\underline{z}$: Disturbance

Figure 2: Linear controller with state and disturbance observers.

is used to reconstruct the disturbance for a disturbance rejection. The actual controller is a state vector feedback controller. Figure 2 shows the generalized structure of the controller for the inverse pendulum that is used in the case study in Section 5. For our example, shown in Section 5, we do not need all the components of this structure. The implemented controller of the inverse pendulum consists of the state feedback $-\underline{R}$ and the observer which is necessary for reconstructing the complete state vector. The disturbance feedforward component was not necessary for the example. In general, the whole controller (gray part of Figure 2) can be expressed by a linear time-invariant state system ((1) and (2)).

The state-space approach is a unified method for modeling and analyzing linear time-invariant control systems. The equations are divided into two parts: a system of (1) relates the state variables $\underline{x}$ and the input signals $\underline{u}$. A second system of (2) relates the state variables $\underline{x}$ and the current input $\underline{u}$ to the output signals $\underline{y}$. The general form of the state-space equations is

$$\dot{\underline{x}} = \underline{A}\underline{x} + \underline{B}\underline{u}, \tag{1}$$

$$\underline{y} = \underline{C}\underline{x} + \underline{D}\underline{u}. \tag{2}$$

Numerical processing

A common method for the realization of digital control systems is now to (a) transform the differential equations into difference equations and (b) convert the variables and parameters from the floating-point to fixed-point or integer numbers. The differential equations (1) and (2) are transformed into a system of recursive difference state equations (*time discretization*)

$$\begin{aligned} \underline{x}(k+1) &= \underline{A}_d\underline{x}(k) + \underline{B}_d\underline{u}(k), \\ \underline{y}(k) &= \underline{C}_d\underline{x}(k) + \underline{D}_d\underline{u}(k). \end{aligned} \tag{3}$$

Now the state and the output signals are represented by the sequences $\{\underline{x}(k)\}$ and $\{\underline{y}(k)\}$.

Numerical integration methods like implicit rectangular or trapezoidal integration are thereby widely used to transform controllers from continuous time to discrete time. With an implicit rectangular integration method, the following equations represent the transformed matrices, where T_s is the discrete sample time and I is the identity matrix:

$$\begin{aligned} \underline{A}_d &= \left[\underline{I} - (\underline{A} \cdot T_s)\right]^{-1}, \\ \underline{B}_{d1} &= \underline{A}_d \cdot T_s \cdot \underline{B}, \\ \underline{B}_d &= \underline{A}_d \cdot \underline{B}_{d1}, \\ \underline{C}_d &= \underline{C}, \\ \underline{D}_d &= (\underline{C} \cdot \underline{B}_{d1}) + \underline{D}. \end{aligned} \tag{4}$$

Obviously matrix $\underline{C}$ remains unaltered whereas $\underline{A}$, $\underline{B}$, and $\underline{D}$ change during the transformation process. Up to now, we have been using floating-point variables. The next step will be to scale the control system (*scaling*) so that the inputs, states, and outputs fit a given numerical range. For determining the minimum and maximum values of the controller state vector $\underline{x}$, it is necessary to run simulations with worst-case controller excitations. The minimum and maximum values of the controller inputs and outputs can be found more easily because they are always defined by controller output limitations (for outputs) and sensor signal ranges (for inputs).

When using implicit rectangular or trapezoidal integration methods, we have to take into account that the matrices $\underline{A}$, $\underline{B}$, $\underline{C}$, and $\underline{D}$ as well as the state vector $\underline{x}$ are transformed (4). For scaling, the minimum and maximum values of $\underline{x}$ must be transformed as well:

$$\underline{x}_D^{\max;\min} = \underline{x}^{\max;\min} + \underline{A}_d \cdot T_S \cdot \underline{B}. \tag{5}$$

Assume we have signed numbers and a numerical range ($\text{Range}_{\text{Num}}$) symmetric to zero. To avoid a range overflow during *multiplication* of two numbers, each variable is scaled to the smaller range $\text{Range}_{\text{Mult}}$ defined as

$$\begin{aligned} \text{Range}_{\text{Mult}} &= \left(\sqrt{\frac{\text{Range}_{\text{Num}}}{2}}\right) - \left(-\sqrt{\frac{\text{Range}_{\text{Num}}}{2}}\right) \\ &= 2 \cdot \sqrt{\frac{\text{Range}_{\text{Num}}}{2}}. \end{aligned} \tag{6}$$

Additionally, the so-called Headroom (in percent) for each variable can be defined. Together with the physical ranges PhyRange, the number range $\text{Range}_{\text{Mult}}$ (6), and the Headroom, the scaling factor s_i for each element of $\underline{x}_d$, $\underline{y}$, and $\underline{u}$ variables can be computed:

$$s_i = \frac{\text{PhyRange}_i}{\text{Range}_{\text{Mult}} \cdot (1 - (0,01 \cdot \text{Headroom}))}. \tag{7}$$

Let $\underline{S} = \text{diag}(s_i)$ be the diagonal matrices composed of the scaling factors s_i. With these scaling matrices, the new discrete and scaled system matrices are as follows:

$$\begin{aligned} \underline{A}_{s,d} &= \underline{S}_{x_d}^{-1} \cdot \underline{A}_d \cdot \underline{S}_{x_d}, \\ \underline{B}_{s,d} &= \underline{S}_{x_d}^{-1} \cdot \underline{B}_d \cdot \underline{S}_u, \\ \underline{C}_{s,d} &= \underline{S}_y^{-1} \cdot \underline{C}_d \cdot \underline{S}_{x_d}, \\ \underline{D}_{s,d} &= \underline{S}_y^{-1} \cdot \underline{D}_d \cdot \underline{S}_u. \end{aligned} \tag{8}$$

The scaling of the matrices with $\underline{S}$ is necessary since input, output, and state vectors are also scaled with $\underline{S}$. Nevertheless, the coefficients of the matrices $\underline{A}_{s,d}$, $\underline{B}_{s,d}$, $\underline{C}_{s,d}$, and $\underline{D}_{s,d}$ could be out of the selected number range because only the ranges of the inputs, outputs, and states were taken into consideration until now. To avoid overflow, each equation has to be prepared to allow the representation of the coefficients within RangeMult. For this, one uses bit shifting operations to allow an efficient implementation of multiplications. Right shifting causes reduced precision with the controller evaluation. So the choice of the word length employed with arithmetic operations is closely related to the shift amount (Shift_{AB}, Shift_{CD}):

$$\begin{aligned} \underline{A}'_{s,d} &= 2^{\text{Shift}_{AB}} \cdot \underline{A}_{s,d}, \\ \underline{B}'_{s,d} &= 2^{\text{Shift}_{AB}} \cdot \underline{B}_{s,d}, \\ \underline{C}'_{s,d} &= 2^{\text{Shift}_{CD}} \cdot \underline{C}_{s,d}, \\ \underline{D}'_{s,d} &= 2^{\text{Shift}_{CD}} \cdot \underline{D}_{s,d}. \end{aligned} \tag{9}$$

The right shift operation leads to the new matrices $\underline{A}'_{s,d}$, $\underline{B}'_{s,d}$, $\underline{C}'_{s,d}$, and $\underline{D}'_{s,d}$. Since the matrices contain only fixed values, shifting must be done *only once* and guarantees that no overflows will occur during computations. To obtain correct values, the computation results must be corrected by a final left shift operation (note that Shift_{AB} and Shift_{CD} are negative)

$$\underline{x}(k+1) = 2^{-\text{Shift}_{AB}} \cdot \left(\underline{A}'_{s,d} \cdot \underline{x}(k) + \underline{B}'_{s,d} \cdot \underline{u}(k)\right), \tag{10}$$

$$\underline{y}(k) = 2^{-\text{Shift}_{CD}} \cdot \left(\underline{C}'_{s,d} \cdot \underline{x}(k) + \underline{D}'_{s,d} \cdot \underline{u}(k)\right), \tag{11}$$

$$\underline{x}(k) = \underline{x}(k+1). \tag{12}$$

The choice of the word length is a compromise between the numerical precision of the controller and the hardware resources required for the implementation. It is useful to provide different word lengths for states, inputs, outputs, and internal multiplication/addition registers. Before hardware synthesis, our approach provides a simulation-based selection of the number of bits for the controller variables before starting the target-specific synthesis of the controller. For the modeling and simulation of scaled state-space controllers, we designed a component for our existing simulation environment CAMeL (Computer-Aided Mechatronics Laboratory) [7], with a word length that is tunable during runtime.

3. AUTOMATED DESIGN FLOW

In this section, we give a brief description of our design flow for automatically implementing digital linear controller systems in hardware. The overall design flow is shown in Figure 3. After modeling the control path mathematically, an analysis and simulation is performed. On the basis of this result, we design the model of the controller. The complete control loop is then simulated. These steps are aided by the tool CAMeL. Up to now, our model is continuous, so the next step is discretization. This is automatically done by an algorithm performing implicit rectangular or trapezoidal integration (4). Since floating-point logic leads to very complex hardware, we scale all variables to a fixed-point range (Section 2). The scaling factors can be determined by simulation with CAMeL or analytical methods [7]. Based on the scaling factors and the not-scaled matrices $\underline{A}_d$, $\underline{B}_d$, $\underline{C}_d$, and $\underline{D}_d$, the scaled matrices $\underline{A}'_{s,d}$, $\underline{B}'_{s,d}$, $\underline{C}'_{s,d}$, and $\underline{D}'_{s,d}$ are automatically computed by a small C-program. After this, the program generates a VHDL package which defines the constants and data types used for the application. This package is included by a parameterizable and generic VHDL template shown in Figure 4. This description can be synthesized by standard synthesis tools to generate the FPGA bit stream to perform the solving of (10), (11), and (12). Thus, after

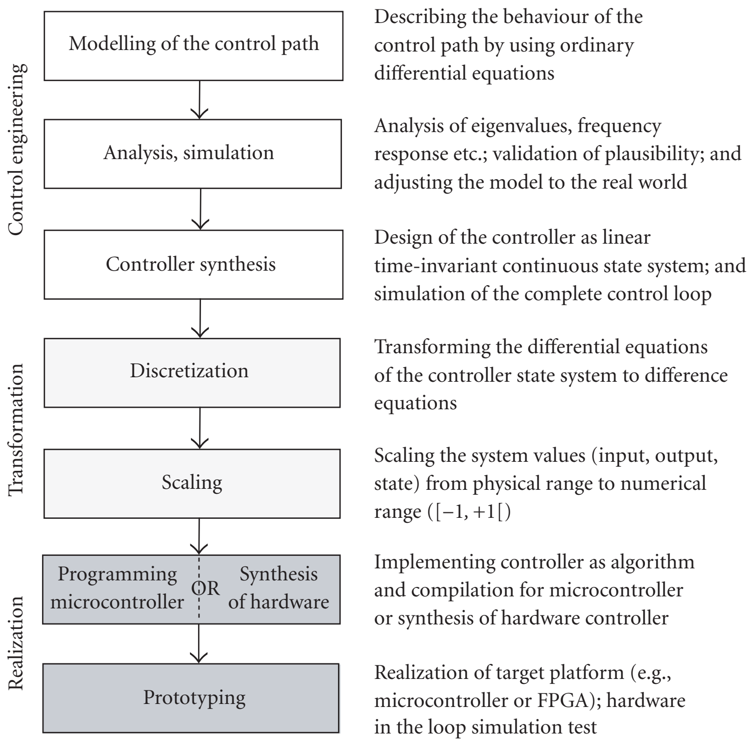

Control engineering
Modelling of the control path
Describing the behaviour of the control path by using ordinary differential equations
Analysis, simulation
Analysis of eigenvalues, frequency response etc.; validation of plausibility; and adjusting the model to the real world
Controller synthesis
Design of the controller as linear time-invariant continuous state system; and simulation of the complete control loop
Transformation
Discretization
Transforming the differential equations of the controller state system to difference equations
Scaling
Scaling the system values (input, output, state) from physical range to numerical range ([−1, +1[)
Realization
Programming microcontroller
OR
Synthesis of hardware
Implementing controller as algorithm and compilation for microcontroller or synthesis of hardware controller
Prototyping
Realization of target platform (e.g., microcontroller or FPGA); hardware in the loop simulation test

Figure 3: Design flow.

determining the scaling factors, the design flow down to the hardware is fully automatic.

4. IMPLEMENTATION OF LINEAR CONTROL SYSTEMS ON RECONFIGURABLE HARDWARE

We compare two different implementations of digital control systems: a hardware controller and a software program running on a microprocessor. To prototype the system, an Aptix System Explorer (http://www.aptix.com/products/mp3.htm) with a Xilinx Virtex FPGA module (XCV2000E, [8]) is used. The FPGA is connected to the control path via a D/A converter and signal transducers and can be configured either for the hardware or for the software solution.

4.1. Hardware implementation

The task of the controller hardware is to compute (10), (11), and (12). Here, $\underline{x}(k)$, $\underline{x}(k+1)$, $\underline{u}(k)$, and $\underline{y}(k)$ are vectors and $\underline{A}_{s,d}$, $\underline{B}_{s,d}$, $\underline{C}_{s,d}$, and $\underline{D}_{s,d}$ are the matrices obtained after discretization and scaling. All matrix and vector elements are fixed-point values. Since both (10) and (12) have exactly the same structure, they can be computed in parallel on two identical units called MECs (matrix equation calculators). Each equation is computed once per sample period which is an integral multiple of the clock period.

The top-level structure of our linear controller design is shown in Figure 4. Besides the MECs, we have two vector registers, one for the controller state (REG x) and one for the output (REG y). The *cycle timer* is a local state machine for synchronizing the MECs.

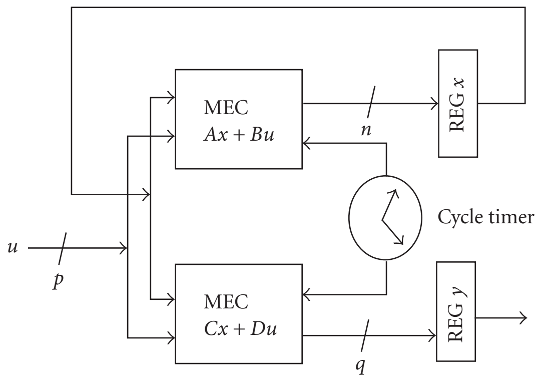


Figure 4: Architecture of the controller hardware.

The MEC components are identical and compute equations of the general form

$$\underline{c} = \underline{M}\,\underline{a} + \underline{N}\,\underline{b} \tag{13}$$

with $\underline{N}$ and $\underline{M}$ matrices and $\underline{a}$, $\underline{b}$, and $\underline{c}$ vectors. Internally, an MEC (Figure 5) consists of a vector adder and two scalar multipliers, each of which computes a matrix-vector product as a sequence of scalar multiplications of the form

$$c = \underline{a}\underline{b} = \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} = a_1 \cdot b_1 + \cdots + a_n \cdot b_n. \tag{14}$$

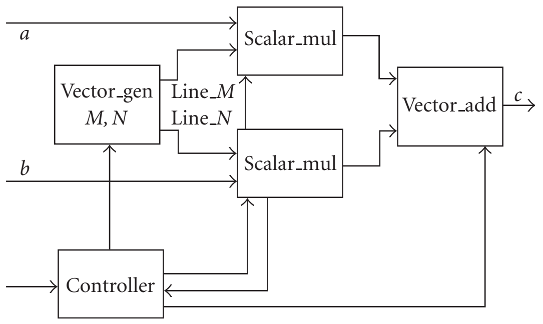


Figure 5: Architecture of the MEC unit.

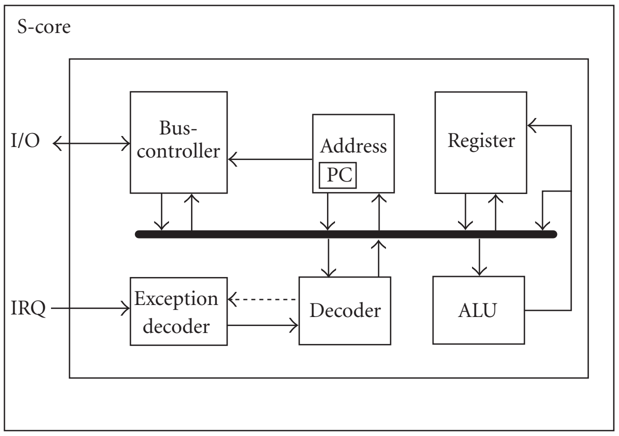


Figure 6: Architecture of the S-core RISC processor [9].

Each scalar multiplier in turn consists of a number of booth-style integer multipliers. The matrices $\underline{M}$ and $\underline{N}$ are constant and hard coded in the *vector_gen* unit which provides the matrices line by line to the scalar multipliers. The design is completely specified in VHDL and parameterizable with respect to the parameters p, n, q, and the word length, where p is the dimension of the input vector $\underline{u}$, n the number of controller states, and q the dimension of the output vector $\underline{y}$. The resource usage of our sample implementation is discussed in Section 5.2.

4.2. Software implementation

The software implementation is based on the S-core microprocessor [9] (Figure 6). The S-core processor design is code-compatible with the Motorola M-core M200 design [10]. It is a 32-bit single-address RISC machine with load/store architecture and a performance of up to 50 MIPS. The processor is available as VHDL core and can be implemented in different silicon technologies. For the case study in this paper, it is synthesized for the Xilinx Virtex FPGA family and an Infineon CMOS gate array technology. Programming of the S-core is supported by the GNU C/C++ tools of the M-core.

5. INVERSE PENDULUM: AN APPLICATION STUDY

5.1. Experiment

Using the design flow presented in Section 3 and the hardware structure proposed in Section 4, we have implemented an FPGA-based linear controller for an inverse pendulum.

The mechanical construction of the pendulum is shown in Figure 7 and the physical model is given in Figure 8. A crab is mounted on a spindle which is rotated by a precision motor. The speed of the motor is simply voltage-controlled. The pendulum mounted on the crab can swing around by 360 degrees. The spindle as well as the axis where the pendulum is mounted on are connected to incremental transmitters which generate pulses if the spindle rotates or the pendulum moves. These pulses are used for determining the crab position (related to a zero position) and the angle of the pendulum. The task of the linear controller is to balance the pendulum up-side-down over the crab, even if the pendulum balance is interfered with mechanical pulses. The physical model (Figure 8) is used to find the parameters for the mathematical model. The parameter d describes the fraction of the mechanical components, m_G and K describe the masses of the parts of the mechanical construction, and F_K is the force which is given by the DC motor to the spindle.

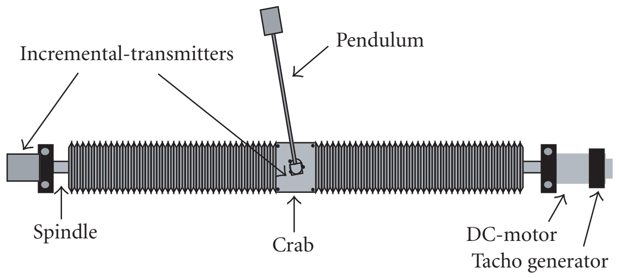


Figure 7: Case study: mechanical construction of the pendulum.

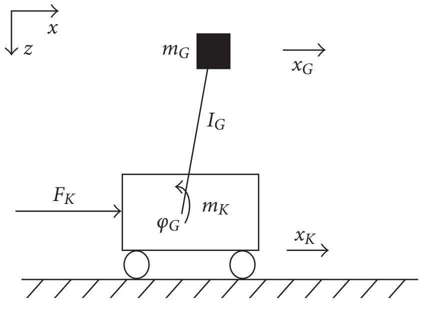


Figure 8: Case study: mechanical model.

The mathematical model of the control path is given by the following equations:

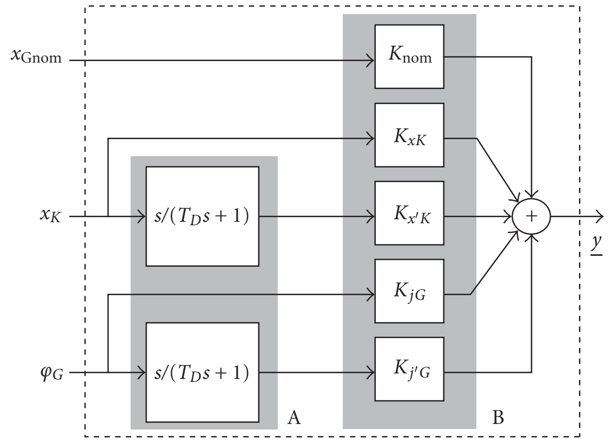


Figure 9: Controller structure.

$$\ddot{x}_K = -\frac{d_K}{m_K}\cdot\dot{x}_K + \frac{m_G\cdot g}{m_K} + \frac{F_K}{m_K},$$
$$\ddot{\varphi}_G = -\frac{d_G}{m_G}\cdot\dot{\varphi}_G + \frac{d_K}{m_K\cdot l_G}\cdot\dot{x}_K - \frac{(m_G+m_K)}{m_K\cdot l_G} - \frac{F_K}{m_K\cdot l_G}. \tag{15}$$

Transforming these equations to the general form

$$\underline{\dot{x}} = \underline{A}\underline{x} + \underline{B}\underline{u} \tag{16}$$

leads to the matrices

$$\underline{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\dfrac{d_K}{m_K} & \dfrac{m_G g}{m_K} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{d_K}{m_K l_G} & -\dfrac{(m_G+m_K)g}{m_K l_G} & -\dfrac{d_G}{m_G} \end{bmatrix}, \quad \underline{B} = \begin{bmatrix} 0 \\ \dfrac{1}{m_K} \\ 0 \\ -\dfrac{d_K}{m_K l_G} \end{bmatrix}. \tag{17}$$

With the state vector $\underline{x} = [x_K\ \ \dot{x}_K\ \ \varphi_G\ \ \dot{\varphi}_G]$ and the vector $\underline{u} = [F_K]$, the mathematical model of the control path is complete.

Figure 9 illustrates the structure of the controller specified in Section 2. Compared with Figure 2 in Section 2, the component A of Figure 9 represents a primitive observer. The differentiators (in A) are necessary to regenerate the state vector. Component B corresponds with the controller-$\underline{R}$ in Figure 2 and realizes the state controller. For the implementation, this representation must be transformed into the state space representation (matrices $\underline{A}$, $\underline{B}$, $\underline{C}$, and $\underline{D}$).

Using the representation from (3) for controller design, we obtain the following controller parameters after discretization (clock rate 1 millisecond):

$$\underline{A}_d = \begin{bmatrix} 0.88176 & 0 \\ 0 & 0.88176 \end{bmatrix}, \quad \underline{B}_d = \begin{bmatrix} 0 & 0.11125 & 0 \\ 0 & 0 & 0.11125 \end{bmatrix}, \quad \underline{C}_d = \begin{bmatrix} -1382304 & 451134 \end{bmatrix}, \quad \underline{D}_d = \begin{bmatrix} 26860 & 1327446 & 447044 \end{bmatrix}. \tag{18}$$

The *vector_gen* units in the MECs (Section 4.1) contain these matrix parameters (after scaling) as hard coded constants. Thus, the VHDL code for the *vector_gen* units is automatically generated from the control path model.

Table 1: Comparison between software and hardware implementation.

Software implementation
- Code size: $8n^2 + 10n + 77 = 129$ Byte
- Clocks per sample = $24n^2 + 14n + 95 = 219$
- Word length: 32 Bit

Technology FPGA:
- #CLBs Processor: 4345 (35% FPGA Virtex 2000)
- Delay critical path: 80.05 ns
- Max. clock: $f_{\text{sys}} = 12$ MHz
- Max. cycle rate: $f_{\text{sys}}/219 = 54.79$ kHz

Technology infineon gate array:
- Clock: $f_{\text{sys}} = 160$ MHz
- Cycle rate: $f_{\text{sys}}/219 = 730.59$ kHz

Hardware implementation
- #MUL: $2(p+n) = 2(3+2) = 10$
- #CLBs: 1123 (5% FPGA Virtex 2000)
- # of sequential Multiplications: $\max(n, q) = 3$
- Clocks per sample = $18\max(n, q) + 2 = 56$
- Word length: 16 Bit extern /32 Bit intern
- Delay critical path: 12.838 ns
- Max. clock: $f_{\text{sys}} = 77$ MHz
- Max. cycle rate: $f_{\text{sys}}/56 = 1, 3$ MHz

5.2. Results

The entire controller design in hardware requires about 5% of the FPGA's CLB resources and can operate at a maximum clock frequency of 77 MHz. Each sample requires 56 clock cycles resulting in a sample rate of 1.38 MHz (sample period approximately 0.73 microsecond). The S-core processor uses 35% of the FPGA resources, it can be clocked at 12 MHz and allows a sample rate of 54.79 kHz (sample period is 18.25 microsecond). By implementing the S-core as an ASIC, operating frequencies of 160 MHz are possible. With such a system clock, the example application can be run with a sample rate of 730 kHz. As shown in Table 1, the sample period increases quadratically with the problem size in the software implementation but only linearly in the hardware implementation.

The experiment shows clearly the advantages of an implementation of digital linear controllers in reconfigurable hardware for the same flexibility as a software implementation; it is possible to implement larger control systems as in software with the same throughput. By exploiting more parallelism in the MEC units (Section 4.1) (e.g., by using more multipliers), it is possible to increase further the sample rate of the hardware architecture. The implicit parallelism of the reconfigurable hardware allows real-time computation with high sampling rates. This property leads to controllers which are more stable than software controllers. Additionally, it is possible to implement also nonstandard fixed-point number ranges in difference to standard floating-point numbers of software implementations for higher precision.

6. CONCLUSIONS

The paper shows how reconfigurable hardware can be used for the implementation of digital linear controllers that require a high amount of digital signal processing. We have presented a new design flow for automatic synthesis of digital linear controllers from the mathematical description of the control path. Furthermore, the differences between hardware and software solutions and their computational complexity were discussed for an example of an inverse pendulum controller. The paper shows that it is possible to implement application-specific hardware structures with a flexibility comparable to the flexibility of software solutions.

Future work will show that this concept can be used for the implementation of self-adapting systems. We plan to apply the described approach to a real-life example of a mechatronic train control system. This case study will be more complex than in this paper since the following additional technical requirements have to be considered:

(a) How can reconfigurable hardware be used for implementation of safety-critical systems?
(b) Can FPGA implementations perform dynamic switching between different controllers?

In this context, dynamic reconfiguration of FPGA might be of high importance.

ACKNOWLEDGMENT

We would like to thank Aptix Corporation (San Jose, Calif, USA) for the technical support during the prototype implementation of our methodology using the Aptix system Explorer MP3C.

REFERENCES

[1] T. B. Goh, Z. Li, B. M. Chen, T. H. Lee, and T. Huang, "Design and implementation of a hard disk drive servo system using robust and perfect tracking approach," *IEEE Transactions on Control-Systems Technology*, vol. 9, no. 2, pp. 221–233, 2002.

[2] S. Koganezawa and T. Hara, "Development of shear-mode piezoelectric microactuator for precise head positioning," *Fujitsu Scientific & Technical Journal*, vol. 37, no. 2, pp. 212–219, 2001.

[3] H. Toshiyoshi, "Microactuators for hard disk drive head positioning," in *The 30th Seiken Symposium on Micro/Nano Mechatronics*, Komaba, Meguro-ku, Tokyo, Japan, March 2002.

[4] R. Cumplido-Parra, S. R. Jones, R. M. Goodall, F. Mitchell, and S. Bateman, "High performance control system processor," in *Proc. 3rd Workshop on System Design Automation (SDA '00)*, pp. 60–67, Dresden, Germany, March 2000.

[5] K. Danne, "Implementierung digitaler Regelungen in Hardware," Project Thesis (FB14/DATE) (in German), University of Paderborn, Paderborn, Germany, October 2000.

[6] O. Föllinger, *Regelungstechnik*, Hüthig, Heidelberg, Germany, 1994.

[7] M. Hahn and T. Koch, "CAMeL-View—Ein Werkzeug zum integrierten CAD-gestützten Entwurf mechatronischer Systeme," in *Simulation im Maschinenbau, SIM '2000*, Dresden, Germany, 2000.

[8] Xilinx, *Virtex Series Configuration Architecture User Guide*, September 2000.

[9] H. Kalte, D. Langen, E. Vonnahme, A. Brinkmann, and U. Rückert, "Dynamically reconfigurable system-on-programmable-chip," in *Proc. 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing (PDP '02)*, pp. 235–242, Gran Canaria Island, Spain, January 2002.

[10] Motorola. M-Core Reference Manual.

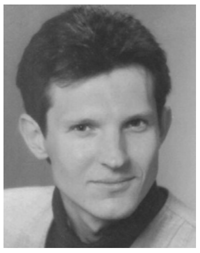
Marcus Bednara received his Diploma degree in computer science in 1998 from the University of Kaiserslautern, Germany. From 1999 to 2002, he was a Researcher and Ph.D. student with the group of Professor J. Teich (Computer Engineering Laboratory) at the University of Paderborn, Germany. Since 2003 he is with the Computer Science Institute of the Friedrich-Alexander University Erlangen-Nuremberg (Hardware-Software-Co-Design group). His research interests are in the area of design automation of VLSI processor arrays, their efficient mapping to reconfigurable architectures, dynamic reconfiguration, and FPGA-based systems for elliptic curve cryptography.

Klaus Danne received his Diploma degree in engineering computing in 2002 from the University of Paderborn, Germany. As a Ph.D. student and member of the Graduiertenkolleg "Automatic Configuration in Open Systems" of the Heinz Nixdorf Institute of Paderborn University, he was a Researcher in the group of Professor J. Teich (Computer Engineering Laboratory) in 2002. Since 2003 he is with the group of Prof. F. Rammig (Design of Parallel Systems). His research interests are reconfigurable computing systems, including partial dynamic reconfiguration, operating system approaches, temporal partitioning, temporal placement, and efficient FPGA implementation of applications such as control systems.

Markus Deppe studied mechanical engineering at the University of Paderborn, Germany. He received his Diploma degree in engineering in 1997. Since then he has been a Research Assistant at the Mechatronics Laboratory Paderborn (MLaP). His research area is the multiobjective parameter optimization combined with distributed real-time simulation of mechatronic systems.

Oliver Oberschelp worked as a trained machine fitter before receiving the university diploma in engineering. After that he studied mechanical engineering at the University of Paderborn, Germany. He received his diploma in 1998. Since then he has been a Research Assistant at the Mechatronics Laboratory Paderborn (MLaP). His research area is design and simulation of mechatronic systems in the context of self-optimizing systems.

Frank Slomka studied electrical engineering and microelectronics at the Technical University of Braunschweig, Germany. After receiving the diploma degree in 1993, he was with the Bosch Telecom. At Bosch, he worked as a software Engineer for digital cordless telephone systems (DECT). From 1996 to 2001, he was with the Rapid Prototyping and Hardware/Software-Co-Design group (Computer Networks and Communication Systems chair), University of Erlangen-Nuremberg. From 2001 to 2002, he was a member of the research staff at the group DATE at the University of Paderborn. Since 2003, he is an Assistant Professor for embedded system design at the University of Oldenburg.



Jürgen Teich received his M.S. degree in 1989 from the University of Kaiserslautern (with honours). From 1989 to 1993, he was a Ph.D. student at the University of Saarland, Saarbrücken, Germany from where he received his Ph.D. degree. In 1994, Dr. Teich joined the DSP design group of Prof. E. A. Lee and D. G. Messerschmitt in the Department of Electrical Engineering and Computer Sciences (EECS) at UC Berkeley where he was working in the Ptolemy project (PostDoc). From 1995 to 1998, he held a position at the Institute of Computer Engineering and Communications Networks Laboratory (TIK) at ETH Zürich, Switzerland, finishing his Habilitation entitled *Synthesis and Optimization of Digital Hardware/Software Systems* in 1996. From 1998 to 2002, he was a Full Professor in the Electrical Engineering and Information Technology Department at the University of Paderborn, holding a chair in computer engineering. Since 2003, he is a Full Professor in the Computer Science Institute of the Friedrich-Alexander University Erlangen-Nuremberg holding a chair in Hardware-Software-Co-Design group. Dr. Teich has been a member of multiple program committees of well-known conferences and workshops. He is a member of the IEEE and an author of a textbook on codesign edited by Springer in 1997. His research interests are massive parallelism, embedded systems, codesign, and computer architecture.