RAPID PROTOTYPING FOR MODEL AND CONTROLLER IMPLEMENTATION

Peter Schwarz
Fraunhofer Institute for Integrated Circuits IIS, Design Automation Division EAS Dresden, Germany

Jörg Uhlig
Institute of Automation and Computer Control, Ruhr-University Bochum, Germany

Keywords: Controller design, prototype, rapid prototyping, fast implementation, model implementation, simulation acceleration, hardware-in-the-loop, HIL simulation, real-time simulation, FPGA, code generation, application-specific hardware, bypass technique.

Contents

Summary

Rapid prototyping is a technique for fast, mostly automated realization of new systems. The development of a small-scale prototype or a prototype which is not yet realized in the final manufacturing technology allows us to test important features and the correct principal functionality of the system already in early design phases. *Controller prototypes* may be realized in software or as application-specific hardware. Code generators and digital synthesis programs are the most important tools to reduce the prototype design effort and are embedded in hardware and software development platforms. This prototyping technology may also be used in computer-aided generation of hardware implementations of *simulation models* for real-time simulation, hardware-in-the-loop simulation, and acceleration of simulation.

1. Definition of Rapid Prototyping

A **prototype** (from the Greek word *πρωτος*, "the first") is a preliminary implementation of a planned product or component which has the principal functionality of the final product; sometimes it is also called a “model” of the new system. **Rapid prototyping** is

the computer-aided or automated generation of a prototype in an early design phase. The prototype has to represent the essential properties but can be realized in another way as the final product. In our context, **new concepts of hardware- and software-based control systems** may be realized and tested very fast by rapid prototyping. Progress in rapid prototyping during the last years leads to the fact that sometimes the automatically generated prototype is so efficient that after testing it may be used as the final implementation.

The rapid prototyping technology may be also used for an **implementation of simulation models** to accelerate the simulation speed or to verify real-time simulation or hardware-in-the-loop simulation.

2. Goals

The very high costs of hardware and software production are responsible for the development of *faster realization methods*. **Rapid prototyping** is a key approach to solve this problem. It supports the following activities in the design process:

- to detect design errors more early;
- to get indications of possible improvements of the design;
- to give potential users a more concrete model of the intended product;
- to test a user interface in an early design phase;
- to test the effectiveness and appeal of a particular solution;
- to develop a model case or practice exercise that can serve as a template for others;
- to get user feedback and reactions to competing approaches.

The common design methodology and the idea of rapid prototyping may be illustrated by the well-known V diagram of an industrial design process (Figure 1).

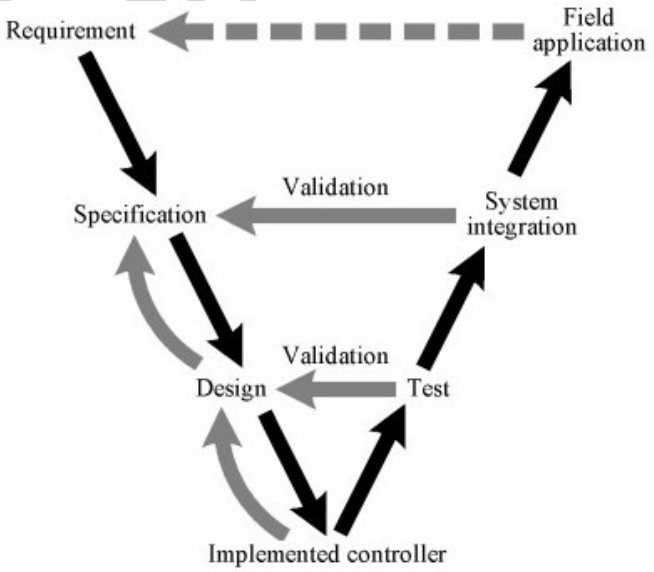

Figure 1: Design process

There exist many slightly different variants of this diagram in the literature if various design tasks, a more or less detailed description of design steps, and special additional aspects are specified by the design engineer. The figure is here adapted to the design of **control systems** (especially of **controllers**) and provides only the essential design steps which are discussed in this article in relation to rapid prototyping. Only a few iteration loops for design validation and improving are presented in this figure.

The different types of controller implementation have to be considered with respect to their specific design problems. The **controller** may be designed either as **software** (SW) running on various types of processors, or as dedicated application-specific **hardware** (HW), or both together which leads to HW-SW-Codesign. Typical controller implementations are as following:

- Program running on a host computer (PC) coupled with the process
- Embedded systems:
  - Microprocessor
  - Programmable controller
  - Digital signal processor (DSP)
  - Microcontroller
- Programmable Logic Controller (PLC)
- Application-specific hardware:
  - Application-specific integrated circuit (ASIC), or especially:
  - Field Programmable Gate Arrays (FPGA)

  mounted on a printed circuit board (PCB) together with other standard components.

One of the most convenient way to develop a new controller algorithm and its prototype implementation is **software running on a host computer** (e.g., a Personal Computer) coupled with the process. This approach is supported by evaluation boards which are plugged into the host computer and may contain the prototype processor, interface devices such as analog-digital and digital-analog converters (ADC and DAC), compiler and debugger (development tool kit, DTK). For industrial applications, controllers are very often realized as **embedded systems** which incorporate *programmable computers* or - more specific - *programmable controllers*. A more detailed description of embedded system is: "Hardware and software which forms a component of some larger system and which is expected to function without human intervention. A typical embedded system consists of a single-board microcomputer with software in ROM, which starts running some special purpose application programs as soon as it is turned on and will not stop until it is turned off (if ever). It will not usually have any of the normal peripherals such as a keyboard, monitor, serial connections, mass storage, etc. or any kind of user interface software unless these are required by the overall system of which it is a part. Often it must provide real-time response.".

A **programmable logic controller, PLC**, is a dedicated computer system mostly used for the automation in process or production engineering. The PLC is a microprocessor-based device with interfaces to the industrial process and is used to monitor the status of the sensors and to calculate output values which are send to the actuators. The term "logic" is used because programming is primarily concerned with

implementing logic and switching operations. The programming of PLCs was restricted to proprietary input languages of the vendors for a long time. Therefore, standardization effort has been spent to overcome these difficulties which results in the standards IEC 1131 and IEC 1499. IEC 1131 specifies the syntax, semantics, and display for a suite of PLC programming languages:

- LD Ladder diagram
- SFC Sequential Function Charts
- FBD Function Block Diagram
- ST Structured Text
- IL Instruction List

One of the essential benefits of this IEC standard is that it allows multiple languages (also in textual or graphical form) to be used within the same programmable controller. This allows the program developer to choose the language best suited to each particular task. The standard IEC 1499 (function blocks for industrial-process measurement and control systems) is directed to decentralized control and distributed architectures. It uses concepts of object-oriented programming, e.g. design patterns, and supports dynamic reconfiguration and re-use of large, complex software modules. Today, PLCs are not often used in rapid prototyping but are essential devices for the final controller implementation.

Controllers may be also realized as **application-specific hardware,** implemented as ASIC (Application-Specific Integrated Circuit) or FPGA (Field Programmable Gate Array). This implementation is necessary if the computer performance is not high enough for signal processing or control algorithms. It is well-known that dedicated HW is 10 to 100 times faster than general-purpose processors due to its high degree of parallel operating devices integrated on the chip.

The focus in this article is on FPGA because the effort to design ASICs is mostly too large to be invested in controller design. FPGAs are integrated circuits or "chips" that can be configured in the "field". FPGA may consist of many millions of transistors and may realize, therefore, also millions of basic digital functions (AND, OR, NAND, NOR, …) or many of more complex functions (ADD, MULT, DIV for integer or floating-point numbers). High-speed controllers may be implemented with these devices and - as a result of their programmability - may be used in rapid prototyping.

With respect to *prototyping*, all kinds of *programmable* devices are candidates for implementation. In our context, **new concepts of hardware and software based control systems** may be realized and tested very fast by rapid prototyping. The first system-validation is not longer done with the complete final implementation of the new system as shown in Figure 1.

Instead a **preliminary version** for the first system validation is realized as a **prototype** of the final product in an intermediate design step. It may be designed quickly and may be changed after evaluation and detection of design errors (Figure 2).

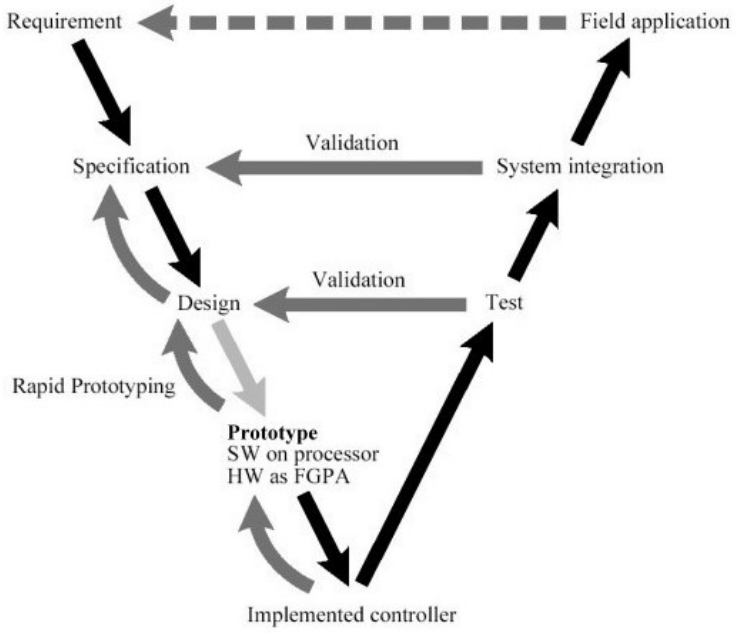

Figure 2: Design process including rapid prototyping

This intermediate activity reduces the design process because

- the prototype implementation may be realized much faster than the final implementation;
- the evaluation of the prototype leads to an early detection of design errors and to design improvements;
- the prototype may be redesigned within a short time;
- the prototype may be transformed into the final implementation in a straight forward manner.

This interpretation of "rapid prototyping" is common use in many technical disciplines. It reflects the fact that it is very important for first functional tests to have a system implementation which is not yet optimized with respect to the requirements of mass production (e.g., cost effectiveness) but fulfills many other design requirements. Especially in the design of controllers and other embedded systems, there is a trend to use the "preliminary prototype", after correction of design errors and performance improvements, as the *final* implementation

It is possible to do this, if

- the generated prototype fulfills all requirements (e.g., function, timing behavior, user interaction),
- the prototype hardware, the target, is the same as in the intended final implementation.

In our special context, "rapid prototyping" becomes therefore more and more a synonym for a *very fast and automated design style*. Powerful code generators with sophisticated code optimization for all widely used processor types and FPGAs with millions of

elements together with automated logic design tools support this design paradigm shift.

The *common* consideration of the controller and the plant is a very important aspect in the design of the controller, e.g., in model-based controller design and in the simulation-based optimization of the controller's behavior. It is also possible to couple a *real system* (e.g., the plant) with a *model* (e.g., of the controller) for simulation purposes. Some possibilities for the interaction of plant and controller and their models, respectively, are presented in Figure 3, which also reflects a typical controller design flow from the simulation and prototyping point of view.

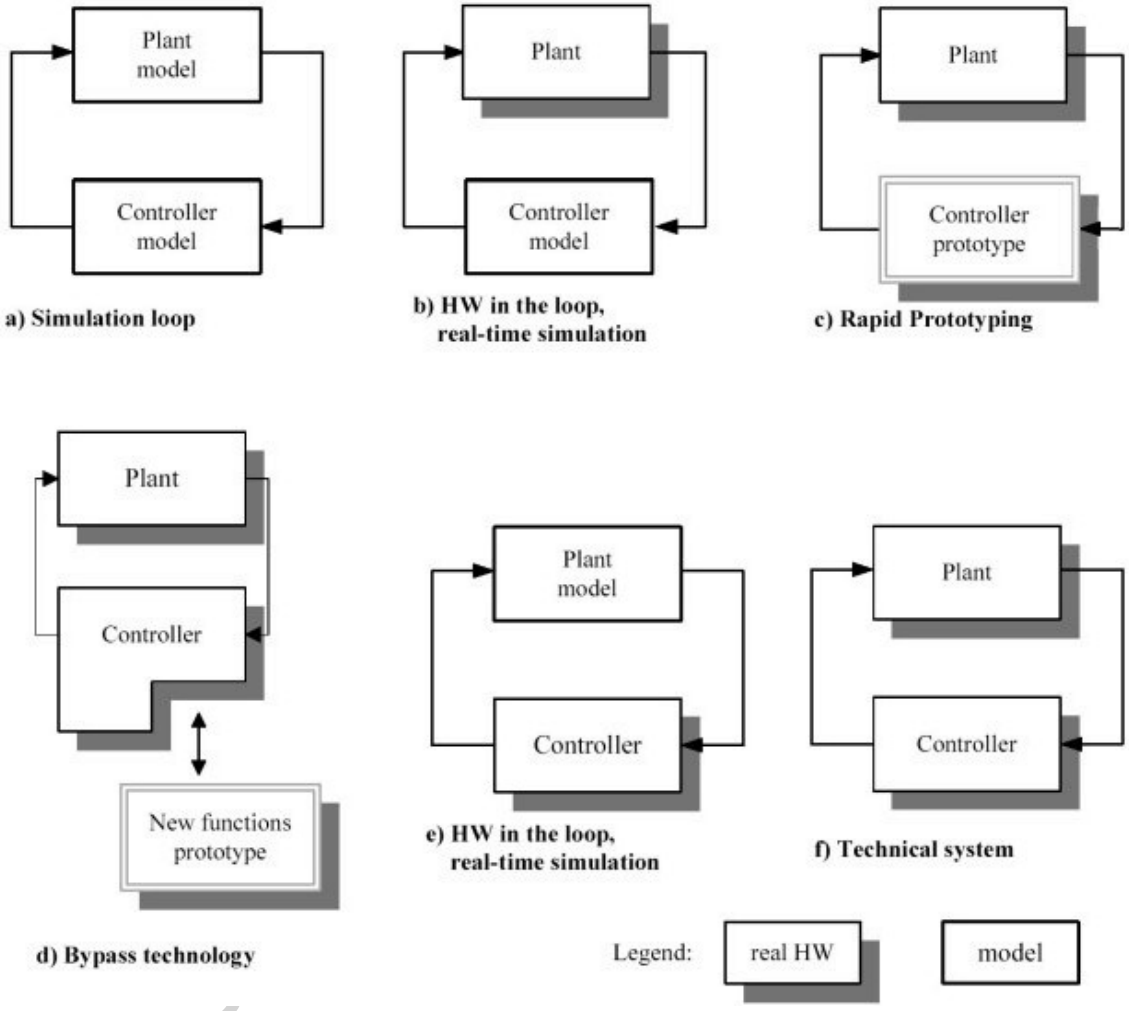

Figure 3: Various forms of interaction between the physical systems (plant, controller, controller prototype) and their models (running on a computer)

The design process starts with the common simulation of the plant *model* and the controller *model* (Figure 3a) which may be realized by one of the well-known simulation tools (e.g., Matlab/Simulink, MatrixX). If high-dimensional physical problems described by differential equations are to be considered, tools as Dymola (with the modeling language Modelica) or VHDL-AMS simulators may also be used.

Hardware-in-the-loop simulation (HIL) and real-time simulation (Figure 3b) are other

important methods for the simulation-based design of control systems and are extensively used in automotive and aircraft design.

The term "hardware-in-the-loop" means: in the "simulation loop" of plant model and controller model (Figure 3a) *one* of these subsystems is substituted by "real hardware" (in Figure 3b, for example, the controller model running in a simulator is coupled with the real plant via appropriate interfaces). The *hardware* is running in real time and, therefore, the *model* has to be simulated fast enough to be coupled with the hardware leading to **real-time simulation**.

In **rapid prototyping**, the controller model is substituted by a preliminary controller implementation on one of the above mentioned HW devices (Figure 3c). It offers the possibility to verify the controller design in a realistic environment so that the final implementation (Figure 3f) based on the same controller design will satisfy the design specifications.

If some parts of existing controllers may be re-used or if the existing controller strategies have to be optimized, the **bypass technology** (Figure 3d) may be applied. It enables the designer to test *new* functions together with the unchanged parts of the remaining controller. E.g., driver software, operating system, or diagnosis subsystems are not changed if some digital signal processing or control algorithms are improved.

Very often the prototype hardware (bypass hardware) does not communicate directly with the system environment via its own I/O interfaces and, therefore, may use the I/O functions of the re-used system residing on the target hardware.

The controller prototype or the final controller implementation may be tested together with the plant model (Figure 3e) - this is another application of HIL and real-time simulation.

In some cases the simulation speed is too low for the computer-aided design and optimization of large or complicated control systems. Special **implementation of models for faster simulation** is therefore necessary (simulation acceleration). The concept of a rapid prototype implementation may be used also in such specialized hardware implementation of model equations for accelerated simulation.

3. General solution

Rapid prototyping of controllers and model implementations for fast simulation follows a design flow, depicted in Figure 4, which is supported by commercial tools, e.g. code generators for software implementation on processors or design tools for Electronic Design Automation (EDA) for hardware implementation, standardized description languages or their graphical representation, and simulators for design verification.

The term "synthesis" is used to emphasize that the design step from detailed specifications to the prototype implementation is heavily supported by algorithms to automate this activity.

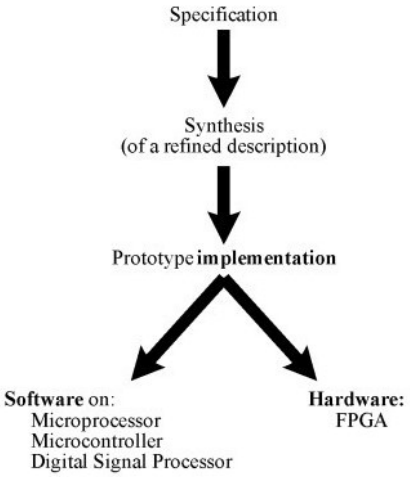

Figure 4: Design of prototypes

Very often, prototypes are running on a different implementation basis as the final system: SW is running on a *host* processor instead of the *target* processor in the final embedded system, HW is realized finally as a FPGA instead of a full-custom or standard-cell based ASIC. Rapid prototyping on HW is related to a lot of scientific problems, as e.g. fully automated design flow; optimization of area, clock frequency and power consumption, but the state-of-the-art for practical applications is only available from the vendors of the FPGA hardware and of the design support tools and the same statement is true for SW prototyping and code generation. Therefore, in many cases the websites of vendors have to be referenced instead of conference papers or journal articles. For the basic HW design tasks, the logic gate and block level design, some books exist which focus on different description languages (VHDL and Verilog which are standardized by IEEE). Some of these books contain simple or more complicated examples of controller designs and therefore allow the not yet very experienced designer the successful implementation of FPGA devices. Rapid SW prototyping with automatic code generation is best described in tutorials or white papers of the vendors.

The **specification phase** is relatively independent of the final implementation in HW or SW. The following formalisms are used to create models as executable specifications:

Discrete systems: State charts and finite state machines (FSM) as representations of automata, discrete algorithms, difference equations, z-domain description.

Continuous systems: Mathematical descriptions in form of

ODE Ordinary differential equations,

DAE Differential-algebraic equations.

Very often a graphical block diagram representation supported by many design tools is used instead of textual formulas. *Linear* systems are mostly described in form of a transfer function, a filter characteristic, or a linear state-space description.

The following **synthesis steps** strongly depend on the implementation target (hardware or software).

-
-
-

Bibliography

Aho A.V., Sethi R., and Ullmann J. D. (1988). *Compilers: principles, techniques and tools*, 769 pp., Reading, MA: Addison-Wesley. [A standard textbook which also covers code optimization.]

Ashenden P. J., Peterson G. D., and Teegarden D. A. (2002). *The System Designer's Guide to VHDL-AMS*, 906 pp., San Francisco: Morgan Kaufmann Publishers. [This book is one of the first comprehensive presentations of VHDL-AMS, a language to model mixed continuous-discrete systems.]

Banerjee P. et al. (2004). Overview of a compiler for synthesizing MATLAB programs onto FPGAs. *IEEE Trans. VLSI*-**12** (2004)3, 312-324. [The paper describes a Matlab-based design flow which leads to the implementation of an algorithm as application-specific hardware.]

Booch G. (1994). *Object Oriented Analysis and Design with Applications*, 608 pp., Reading MA: Benjamin/Cummings, 2nd ed. [A standard book about object oriented software design.]

Dunning G. (2001). *Introduction to Programmable Logic Controllers*, 480 pp., Delmar Thomson Learning; 2nd ed. [An application-oriented textbook on PLC design and application.]

Elmqvist H., Mattsson S.E., and Olsson H. (2002). New methods for hardware-in-the-loop simulation of stiff models. *Proc. Modelica'2002*, Oberpfaffenhofen, Germany, 59-64. (download from http://www.Modelica.org/Conference2002/papers.shtml). [This conference paper presents an approach to real-time simulation in the context of the Modelica language.]

FPGA (2004). 12th ACM *International Symposium on Field-Programmable Gate Arrays*. [An example of yearly organized conferences on FPGA synthesis.]

Gamma E., Helm R., and Johnson R. (1995). *Design Patterns - elements of reusable object-oriented software*, 416 pp., Reading, MA: Addison-Wesley. [A textbook about important aspects of object-oriented software design.]

Hamblen J.O., and Furman M.D (2001). *Rapid Prototyping of Digital Systems - a Tutorial Approach*, 270 pp., Boston: Kluwer. [An introduction to rapid prototyping based on Altera's FPGA design platform.]

Harel D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming* **8**(1987)3, 231-274. [A classical paper on graphical model and system representation.]

Harel D. et al. (1990). Statemate: a working Environment for the Development of Complex Reactive Systems, *IEEE Trans. Software Engineering* **SE-16**, 403-414. [The description of one of the first tools on system simulation based on Statechart representation of dynamical systems.]

Haufe J., Fritsch C., Gulbins M., Lueck V., and Schwarz, P. (1998). Real-time debugging of digital integrated circuits. Paris: *Proc. Design, Automation and Test in Europe Conference, User Forum (DATE'98)*, 235-241. [The paper describes the debugging extensions in FPGA-based rapid prototyping.]

Haufe J., Schwarz P., Berndt T., and Grosse J. (1998). Accelerated Logic Simulation by Using Prototype Boards. Paris: *Proc. Conf. Design, Automation and Test in Europe (DATE'98)*, 183-189. [The paper describes the coupling of logic simulators with FPGA prototyping boards to accelerate digital simulation of large control and data processing algorithms.]

Hughes T.A. (2000). *Programmable Controllers*, 334 pp., Instrument Society of America; 3rd edition. [An introduction to PLC design and application.]

Patterson D. A., and Hennessy J. L. (2003). *Computer Architecture - A Quantitative Approach*, 1100 pp.,

San Francisco, CA: Morgan Kaufmann Publ. [A classical textbook on computer architecture and basics on compiler writing.]

Reinemann T., and Kasper R. (2001): High Speed Implementation of Controllers and Filters for Mechatronic Systems; http://www.techonline.com/community/home/14817. [Link to a report on FPGA prototyping for controller design.]

Salcic Z., and Smailagic A. (1997). *Digital Systems Design and Prototyping Using Field Programmable Logic*, 648 pp., Boston: Kluwer. [Introduction to FPGA based rapid prototyping.]

Smith D. J. (2000). *HDL Chip Design - a Practical Guide for Designing, Synthesizing and Simulating ASICS and FPGAs using VHDL and Verilog*, 448 pp., Madison, AL, Doone Publications, 8. printing. [A widely used book on logic synthesis in the context of FPGA design.]

Wilson B. G., Jonassen D. H., and Cole P. (1993). Cognitive approaches to instructional design. In G. M. Piskurich (Ed.), *The ASTD handbook of instructional technology* (pp. 21.1-21.22), New York: McGraw-Hill. Also available at http://www.cudenver.edu/~bwilson. [A general discussion on prototyping, independent of special applications.]

Instead of publications, Internet site addresses should be used as references to design tools (e.g., simulators and code generators) and prototype platforms as well as to standardized modeling languages. With respect to the rapid development of such software (and the occasionally rapid change of tool names and trade marks), the best way to up-to-date information is to check the homepages of the hardware and software providers:

http://www.dspace.de [dSpace - a vendor of prototyping systems.]

http://www.dynasim.se [Dymola, a multi-physics simulator with Modelica as modeling language.]

http://en.etasgroup.com [ETAS - a vendor of prototyping systems.]

http://www.mathworks.com [Control system design and simulation environment Matlab/Simulink, widely used entry point for rapid prototyping.]

http://www.modelica.org [Modelica is a powerful modeling language; many links to Modelica-related publications and the language specification, also to hardware-in-the-loop and real-time simulation applications.]

http://www.ni.com/matrixx [National Instruments: Control system design and simulation environment MatrixX, also used for rapid prototyping.]

http//:www.synopsys.com [Synopsys: one of the leading vendors of software for synthesis of digital hardware and system simulation.]

http://www.nr.com [Link to Numerical Recipes, a collection of algorithms in numerical mathematics.]

http://www.vhdl.org/analog [This is the official website of the IEEE standardization working group for the powerful mixed-signal modeling language VHDL-AMS, the Analog and Mixed Signal extension of VHDL.]

Rapid prototyping, real-time simulation, and hardware-the-loop are areas with a lot of differently used words, especially in the context of vendor's presentations. It may be helpful to search in some glossaries and dictionaries, e.g.

http://www.hyperdictionary.com/dictionary and http://wikipedia.com or other search machines.

Biographical Sketches

Peter Schwarz received the diploma and the Ph.D. degree in electrical engineering from the Dresden University of Technology in 1964 and 1967, respectively. He worked in the Ro-botron Computer Company in Dresden and was responsible for research and development in a CAD group. From 1982 to 1991 he was the leader of the research group "Simulation" at the Central Institute for Cybernetics and Information Processes of the Academy of Sciences in Dresden. He was engaged in the development of a multi-level, mixed-signal simulator KOSIM which was used in industry and research institutes at that time. He received the Habilitation degree from the Dresden University of Technology in 1989. Since 1992 he has been working with the Fraunhofer Institute for Integrated Circuits, Design Automation Division EAS Dresden.

He is the head of the Modeling and Simulation department with about 30 engineers, mathematicians, and physicists. His special interests are now multi-level, mixed-signal modeling and simulation of complex heterogeneous systems, web-based simulation and optimization, and knowledge transfer in life-long learning. Application areas are integrated circuit design, micro-electrical-mechanical systems (MEMS), telecommunications, mechatronics, and automation systems. He is member of IEEE and VDE.

Jörg Uhlig received the diploma degree in electrical engineering at the Ruhr-Universität Bochum 1996. Currently he is working as a research assistant at the Institute of Automation and Computer Control, Ruhr-Universität Bochum. His current research interests are in control-ling redundant lightweight manipulators. His activities are subdivided in building an open control system and in applying modern control methods to a four link lightweight manipulator. Several adaptive, robust and predictive control methods are used to find the best solution for this manipulator.