

# Rapid controller prototyping with Matlab/Simulink and Linux

Roberto Bucher\*, Silvano Balemi

*Department of Innovative Technologies, University of Applied Sciences of Southern Switzerland (SUPSI), 6928 Lugano-Manno, Switzerland*

Received 13 November 2003; accepted 21 September 2004

## Abstract

This paper presents a Rapid Controller Prototyping System based on Matlab, Simulink and the Real-Time-Workshop toolbox. Executable code is automatically generated for Linux RTAI, a hard real-time extension of the Linux Operating System. The generated code runs as a normal user space hard real-time application on a standard personal computer with the RTAI extension of the Linux Operating System. This environment can be used to quickly implement real-time controllers. Moreover, standard hardware allows the use of rapid prototyping techniques not only during development activities but also during system operation. A didactic and an industrial application are presented which demonstrate the capabilities and the performance of the environment. © 2004 Elsevier Ltd. All rights reserved.

*Keywords:* Control applications; Rapid controller prototyping; Computer-aided control system design; Distributed controls; Education

## 1. Introduction

Rapid controller prototyping techniques allow to implement and validate control strategies during the development process: users can work within the same environment from the requirement analysis to the controller design and implementation phase (see Fig. 1).

A problem with rapid controller prototyping systems is the cost and complexity of the control hardware typically used. Such hardware makes sense only as long as no hardware dedicated to the application is available.

An ideal rapid controller prototyping platform should rely on generic software and hardware components. Then, such a platform could be used from the development process to functional prototypes and even to low-volume products.

The present paper shows how a rapid controller prototyping systems has been created by exploiting

standard x86-based computing platforms provided with Linux software in combination with a commercial Computer-Aided Control System Design (CACSD) software (Matlab/Simulink).

In the next sections, the RTAI (Real Time Application Interface) project and the implications for the current work are presented. Then, the elements composing the rapid controller prototyping system are explained. A didactic and an industrial application demonstrate the capabilities of the tool developed. Finally an outlook of future work is given.

## 2. The RTAI project

### 2.1. A real-time extension to Linux

The work presented in this paper exploits the RTAI extension to the Linux Operating System (OS) proposed by the Dipartimento di Ingegneria Aerospaziale of the Politecnico di Milano (DIAPM) (<http://www.rtai.org>). This extension adds hard real-time capabilities to a Linux OS, allowing sample frequencies up to several thousands of cycles per second with a jitter of just a few

\*Corresponding author. Tel.: +41 91 610 85 48; fax: +41 91 610 85 70.

*E-mail addresses:* [roberto.bucher@supsi.ch](mailto:roberto.bucher@supsi.ch) (R. Bucher), [silvano.balemi@supsi.ch](mailto:silvano.balemi@supsi.ch) (S. Balemi).

*URL:* <http://www.rtai.org>.

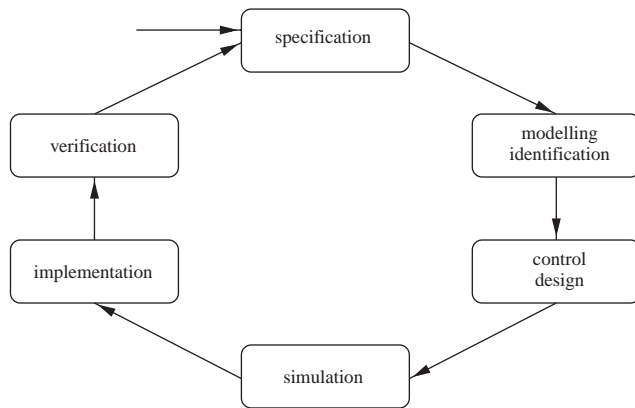


Fig. 1. Control system design phases.

microseconds. Real-time processes can run both in the kernel and in the user area.

The RTAI extension was created as an environment for implementing low cost data acquisition and digital controller systems (Beal et al., 2000) and it is distributed under the GNU General Public License (GPL) for the kernel part and under the GNU Lesser General Public License (LGPL) for the user part (<http://www.gnu.org>).

The RTAI extension has already reached maturity and has been widely exploited in several areas (Bianchi & Dozio, 2000; Bianchi, Dozio, Mantegazza, & Ghiringhelli, 1999a; Bianchi, Dozio, Martini, & Mantegazza, 1999b; Dozio & Mantegazza, 2003a,b). At present, different projects and industrial applications take advantage of Linux RTAI. The Orocos project (<http://www.orocos.org>) aims to develop an open robot control environment for generic robotic devices (manipulators, mobile robots, humanoids, etc.). Another example of industrial application running Linux RTAI is given by the 2D laser cutting machine developed by PLATINO, where the numerical control is based on a standard PC with Linux OS and the RTAI real-time extension ([http://www.primaindustrie.com/pr\\_platino.html](http://www.primaindustrie.com/pr_platino.html)). Still another interesting application is given by the ALMA project (Atacama Large Millimeter Array <http://www.alma.nrao.edu>), where a cluster of 16 nodes running RTAI-patched Linux kernels performs data filtering, data windowing, fast Fourier transforms and phase corrections at a processing rate of approximately 1 GFLOPS.

## 2.2. Linux RTAI and CACSD software

Thanks to the experience from past projects which exploited the CACSD software Matlab/Simulink together with the toolbox Real-Time-Workshop (RTW) from Mathworks (<http://www.mathworks.com>) for creation of executable code for various platforms, the authors started to integrate this CACSD software with Linux RTAI.

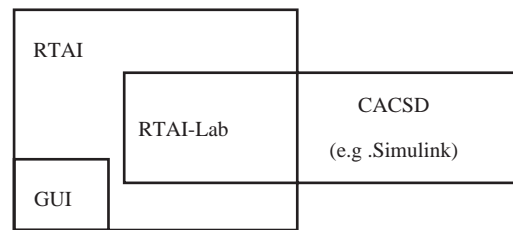


Fig. 2. Integration of RTAI and CACSD software.

Matlab/Simulink is a design and simulation tool used at most universities and numerous industrial companies. Simulink is a graphical environment which allows the user to create models for dynamic systems simply by connecting blocks from available libraries. Among others, some blocks implement linear systems given as transfer functions or state-space realizations both in continuous and discrete time. The RTW toolbox generates C-code from a Simulink model without the need of any specific programming knowledge.

The work by the authors has been later integrated and extended within the framework of the RTAI project. Since version 2.4.11 the RTAI distribution contains a tool called *RTAI-Lab* which provides a structured framework for the integration of RTAI with different CACSD environments (Bucher & Dozio, 2003). The current RTAI-Lab implementation includes support for Matlab/Simulink/RTW from Mathworks and the open source Scilab/Scicos suite (<http://www.scilab.org>). Implementations for other CACSD environments such as the product MatrixX from National Instruments are planned (Fig. 2).

## 3. RTAI-Lab

RTAI-Lab is an open source project which provides an integration between Linux RTAI and several CACSD software products. It is a part of and installed with the RTAI distribution software.

### 3.1. The RTAI-Lab environment

RTAI-Lab relies on the CACSD software for the controller design and for the corresponding code design and generation. The RTAI-Lab tool provides the CACSD environment with some additional specific blocks and building options. Then, the code generated can be used in a RTAI framework, executed in soft or hard real-time and monitored by a generic graphical user interface (GUI). During run-time, the CACSD software used for code generation is no longer needed.

The GUI application for RTAI runs in soft real-time and can be started either on the same PC, where the

hard real-time task is running, or can be started remotely on the LAN. Then, the monitoring task runs on another PC running RTAI and communicates with the hard real-time control task through the RTAI real-time middleware layer, called `net_rpc` (see Fig. 3).

Fig. 4 shows the RTAI-Lab graphical user interface which can be used to monitor different signals and to change parameters while the real-time task is running. The corresponding application is called `xrtailab`.

The current version offers three types of measurement instruments: digital scopes, leds and meters. Each instrument is provided with a window for displaying and changing the instrument layout; another window manages the logging of generic multidimensional data. Target tunable parameters can be displayed and changed on the fly.

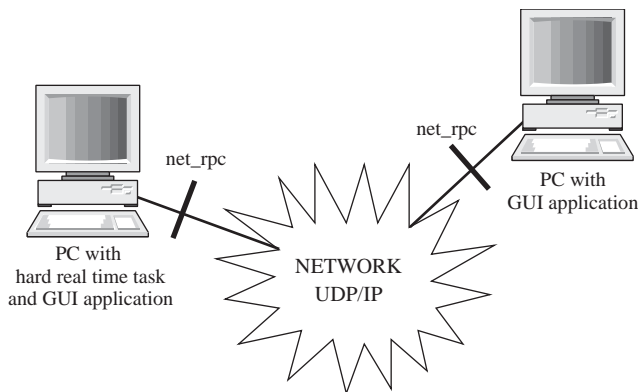


Fig. 3. RTAI-Lab with remote monitoring of a hard real-time control task.

### 3.2. Tool usage: building and running executable code

In order to build and run executable code using the CACSD software Matlab/Simulink, three distinct steps can be identified.

*Model definition:* The designer creates an appropriate Simulink model using blocks from the standard built-in libraries of the CACSD tool and additional interface blocks from the RTAI-Lab library. Typically, a Simulink model used to validate a controller is modified by replacing the plant model with the I/O blocks from the RTAI-Lab library and by substituting the standard scopes with the RTAI-Lab scopes. Then, the integration of sensors and actuators with the controller is accomplished.

*Code generation:* The second step involves the C-code generation. The user must change the target configuration of RTW by introducing the specific names of the three files (System target file, template makefile, make command) needed by RTAI-Lab. Then, the “build model” command in Simulink starts the generation and compilation of the C-code with the help of the three files indicated above. The executable code is obtained.

*Code execution:* Finally, the standalone executable code can be transferred to any target machine running the same version of RTAI used for compilation. There, it can be run without the need of the presence of CACSD software.

Starting the `xrtailab` application on any machine with RTAI allows to monitor the real-time application running on the target machine.

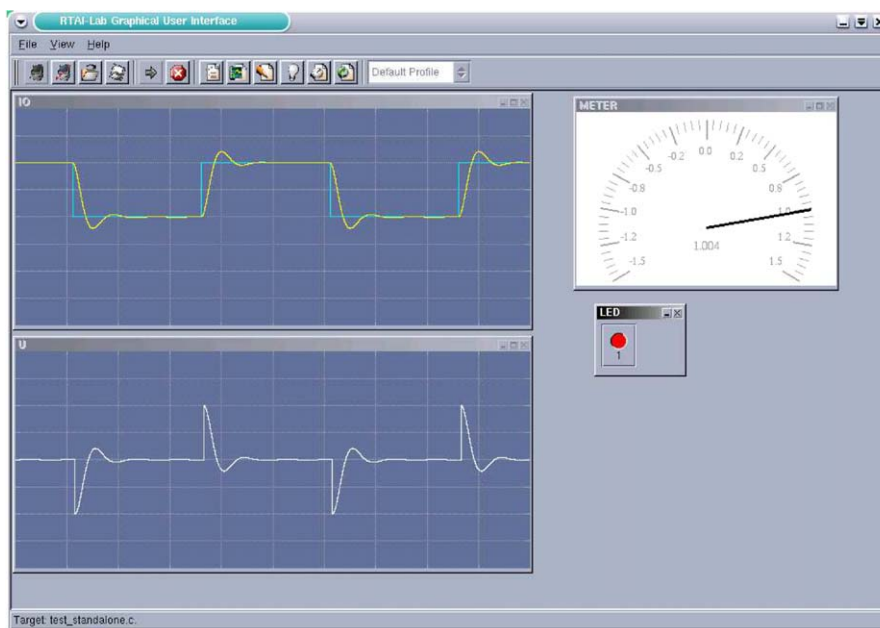


Fig. 4. RTAI-Lab graphical user interface.

### 3.3. Technical issues

The RTAI-Lab tool follows the same scheme found in the native client/server architecture of some CACSD software such as the Matlab external mode. RTAI-Lab is an alternative to the Matlab/Simulink external mode and allows to run the generated code without the need of the CACSD software used to generate the code, thus allowing the replication of the run-time software without any particular constraint (as long as the RTAI extension has been installed).

In RTAI-Lab the communication layer between host and target is fully integrated into the real-time code via net\_rpc without the need to adapt and link it with an ad hoc transport layer implementation.

In order to combine RTAI and Matlab/Simulink, the RTW facilities for integration of different targets have been exploited.

#### 3.3.1. Matlab TLC and template makefile

RTW requires two special files in order to be able to generate code for a specific target:

- the Target Language Compiler (TLC) file used to control the code generation and
- the Template Makefile used to create the project specific Makefile which contains all directives needed to compile and link the C-code generated by RTW.

For the creation of the TLC files the default Target Language Compiler file could be used with little modifications. The resulting Template Makefile supports several Matlab toolboxes (Signal Processing Toolbox, Fuzzy Control Toolbox, etc.) and the integration of the generated code into the RTAI-Lab environment.

Both continuous and discrete-time blocks are supported, thus remarkably increasing the potentiality and flexibility of RTAI-Lab.

#### 3.3.2. Files for I/O boards

In order to interface physical plants, acquisition and interface boards had to be integrated in RTAI-Lab. For each function provided by a DAQ board (AD-conversion, DA-conversion, etc.), two modules have to be programmed:

- A C-MEX S-function to integrate the I/O in the Simulink environment and
- A C-module, linked to the RTAI kernel module, to access the board.

The C-MEX S-function only manages the module parameters (number of inputs, number of outputs, board address, sampling time, etc.). The C-module handles all I/O functions (register programming), for instance:

- writing values to a D/A converter,
- controlling an A/D conversion and reading the resulting value,
- controlling a digital I/O,
- reading the counter of an incremental encoder.

#### 3.3.3. RTAI-Lab library for Simulink

RTAI-Lab uses the COMEDI drivers for the integration of the acquisition boards (<http://www.comedi.org>). The creation of drivers for boards not yet supported by the COMEDI project requires little effort. Fig. 5 shows the current Simulink library. The blocks rtai\_scope, rtai\_meter, rtai\_led and rtai\_log allow the real-time task to exchange data with the GUI application.

#### 3.4. Reasons for using RTAI-Lab

The proposed solution relies on Matlab, Simulink and the RTW toolbox but proposes the use of an additional

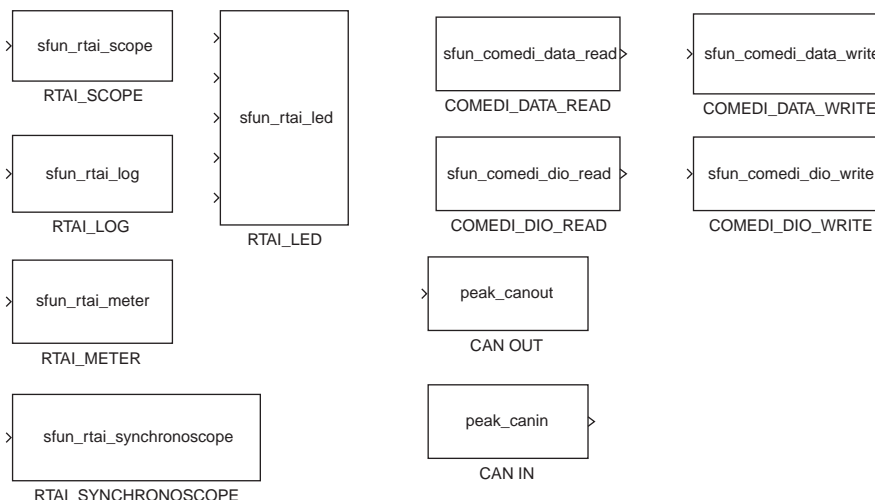


Fig. 5. RTAI specific Simulink library.

environment. The cost of learning this environment is offset by the following advantages:

- Linux RTAI is a free, open source, hard real-time operating system solution. It is *easily customizable and extendable* in function of specific application needs.
- The same PC used for developing the application can act also as controller, thus allowing a *single platform solution*.
- The system *performance can be scaled up* thanks to the possibility of distributed control.
- The real-time application can be easily linked to other soft or hard real-time or even to asynchronous tasks running remotely.
- All runtime code, including the monitoring and parameter update tasks, runs without the Matlab, Simulink and RTW tools used for the code generation.

## 4. Applications

### 4.1. A didactic application: the inverted pendulum

The inverted pendulum of Fig. 6 has been controlled by software generated with the RTAI-Lab environment.

A state-space feedback controller with reduced order observer has been implemented. The Simulink model contains the controller–observer pair in state-space form.

Fig. 7 (top) shows the Simulink model used to simulate the inverted pendulum: The pendulum block contains the pendulum motion equations (see the top diagram of Fig. 8).

Fig. 7 (bottom) instead depicts the model used for generation of the executable code: The pendulum block contains now the S-functions for the interfaces (see the bottom diagram of Fig. 8).

As one can notice, the step from the simulation to the generation of the real-time control code is completely transparent.

#### 4.1.1. Distributed control

The same mechanism used to communicate between the real-time task and the RTAI-Lab GUI can be used to establish a communication between different hard real-time tasks. This makes it possible to create a distributed control system within a single PC or in a LAN, using the `net_rpc` layer and the `rtnet` hard real-time network driver (<http://www.rts.uni-hannover.de/rtnet>, see Fig. 9).

Distributed control for the inverted pendulum has been implemented at the control laboratory at SUPSI. A first PC is responsible for input and output. The Simulink model used to generate the code contains all blocks for the sensors and the actuator needed to interface the system hardware (see Fig. 10).



Fig. 6. Inverted pendulum.

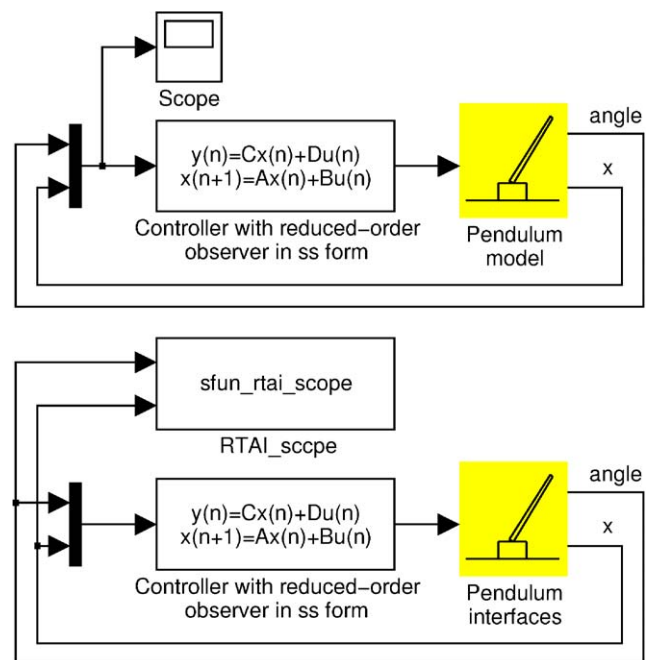


Fig. 7. Simulink model for simulation (top) and code generation (bottom).

A second PC is responsible for the computation of the control law. The Simulink model of the corresponding controller task is shown in Fig. 11.

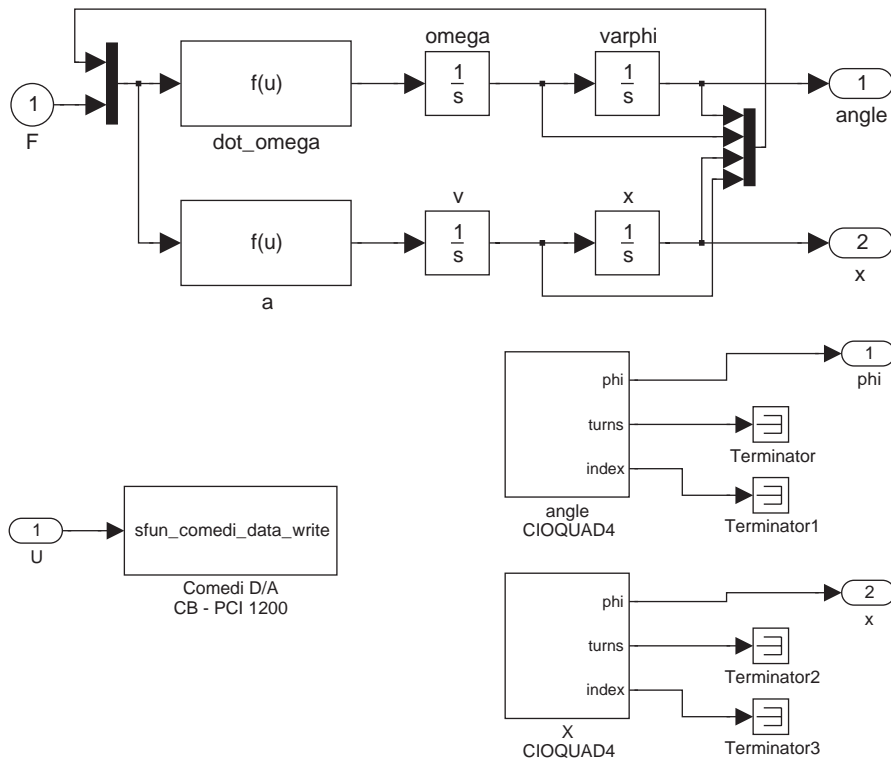


Fig. 8. Pendulum block for the simulation (top) and code generation (bottom).

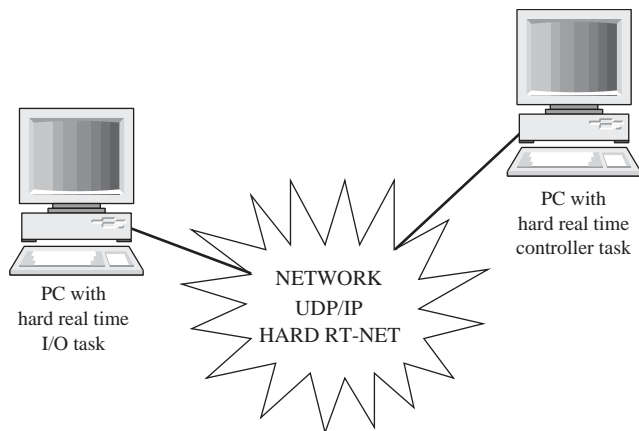


Fig. 9. Distributed control of the inverted pendulum.

The inter-task communication between the hard real-time applications is implemented using a structure provided by RTAI, called *mailbox*.

The values collected from the sensors by the real-time task on the first PC are stored into a local mailbox represented by the *MBX-IO* block in the Simulink scheme. The second PC fetches these values from the mailbox through the LAN by means of the procedures provided by *net\_rpc* and computes the control value. The actuation value is then saved into a second remote mailbox on the first PC (*MBX-CTR* in the Simulink scheme), read by the input/output task and then written to the D/A board.

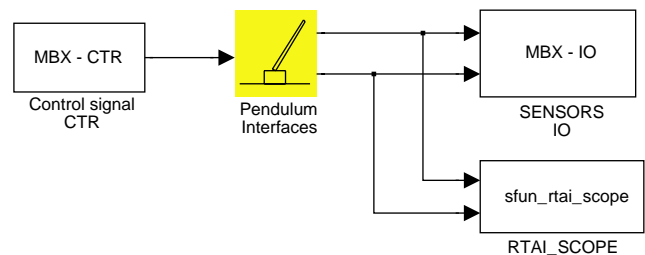


Fig. 10. Simulink model of the IO real-time task.

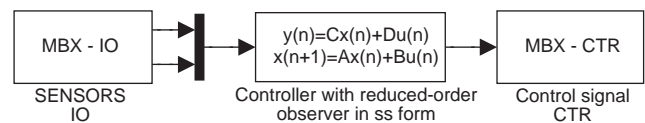


Fig. 11. Simulink model of the controller real-time task.

#### 4.2. An industrial application: a platform for precision positioning

The increasing interest in nanotechnology requires the use of positioning systems of always smaller dimensions and higher bandwidths. An example is the atomic force microscope presented by Balemi et al. (2004), which runs at a sampling frequency of 40 kHz achieved thanks to an ad hoc DSP-based control hardware.

However, such ad hoc control systems can be adapted to new applications only with noticeable effort. Thus, a more flexible solution is desirable, which simplifies the development of a suitable control hardware and of the corresponding program.

Following these thoughts, the development of a PC104-based system has been started. Besides the processor board, single boards are used to interface voice-coil motors, sinusoidal encoders or capacitive sensors. According to the application needs, the hardware can be configured by stacking the necessary number of (also identical) boards on top of each other. The system with interface boards for voice-coil motors and for sinusoidal encoders is shown in Fig. 12.

The choice of a PC104 system with a x86-based processor board allows to exploit the possibilities offered by Linux RTAI and by its interfaces with Matlab/Simulink. The system to be controlled can be identified, based on measurement data uploaded to Matlab/Simulink (see Fig. 13).

Then, the controller can be designed and tested in simulation. Finally, by substituting the simulation model with the block for the interfaces, the executable code can be generated and the real-time control started, while the measurement data can be displayed or recorded for later analysis (see Fig. 14).

In order to minimize the cost of the PC104 platform, the system may have only the minimal outfit (disk space or RAM size) which is necessary to run the real-time code. The compilation of the real-time code can be performed on a standard PC with Linux RTAI connected to the network. The same PC can also be used for the display and for uploading the measured data.

Measurement data indicate sampling frequencies up to 20 kHz with jitter less than 10  $\mu$ s with a 200 MHz 486 processor. The system is completely flexible: a reconfi-

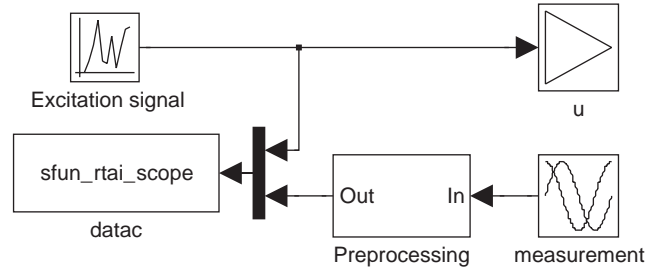


Fig. 13. Model for identification of a high-precision positioning system.

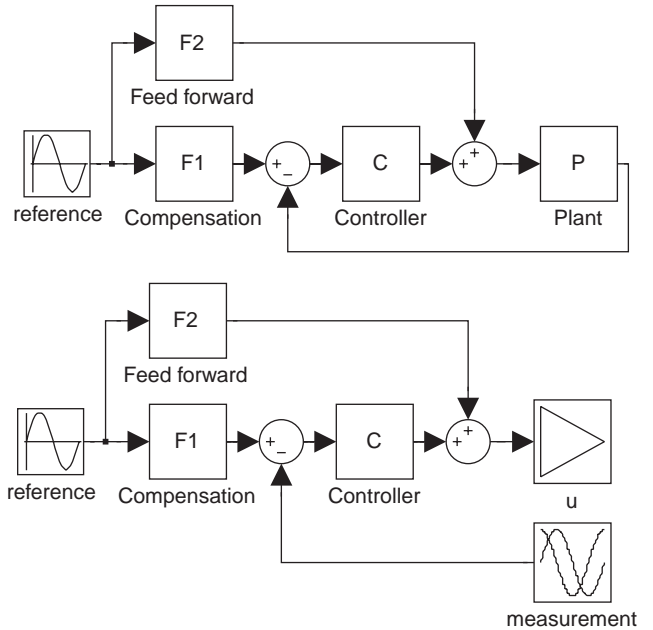


Fig. 14. Simulation (top) and control code generation (bottom) within the same environment.

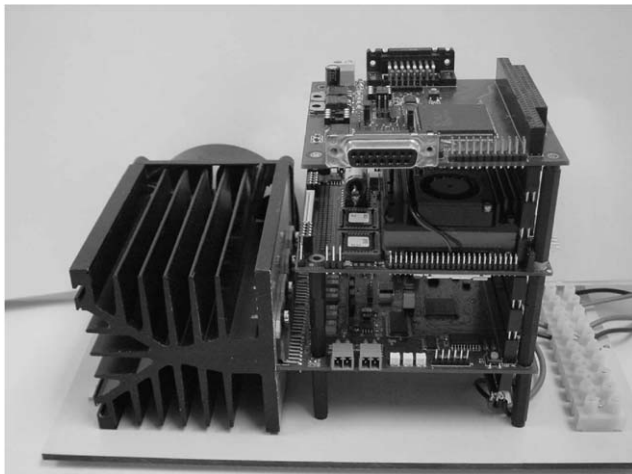


Fig. 12. PC104-based system for high precision applications.

guration just requires stacking the appropriate number of boards, choosing the desired ISA addresses with the available jumpers and inserting the corresponding interface blocks in the Simulink model.

The system will be used not only as a base for further research in nanopositioning and manipulation, but also in commercial low-volume products, thus demonstrating the utility of the Linux RTAI project.

## 5. Other related issues

### 5.1. Kernel 2.6

RTAI and RTAI-Lab has been ported to the Linux kernel 2.6.x. The new release of RTAI Linux is now based on the ADEOS patches from Philippe Gerum

(Adaptive Domain Environment for Operating Systems <http://home.gna.org/adeos/>).

### 5.2. Hardware-in-the-loop simulation

The present work has been extended to hardware-in-the-loop simulation with the project “Multibody Dynamic Analysis Software on real-time distributed systems” (<http://www.aero.polimi.it/~mbdyn/mbdyn-rt>).

This project uses general purpose multibody analysis software for distributed simulation of complex physical systems over real-time networks. Such an approach offers an open-ended scalable development environment which makes it possible to adapt to subsequent growth of the model complexity.

Single multibody subsystems may be analyzed on different machines, whereas each machine can be uniprocessor or symmetric multiprocessor.

The control algorithms can be designed and tested using RTAI-Lab together with Matlab/Simulink and RTW. The mix of hard real-time tasks (control and multibody analysis) and soft real-time tasks (monitoring, animations) can be flexibly adjusted thanks to the `net_rpc` layer described above.

### 5.3. Scilab/Scicos

All applications presented above have been implemented at SUPSI also with the open source CACSD software Scilab/Scicos. Distributed control is possible also with a mix of hard real-time applications generated from Scilab/Scicos and from Matlab/Simulink/RTW.

## 6. Conclusions

A simple and flexible tool for rapid control prototyping has been presented. This tool allows to reach sampling frequencies up to 20 kHz, which is sufficient for controlling most mechanical plants. Using RTAI-Lab, the Matlab/Simulink CACSD software can be used for data acquisition, analysis, design and simulation. Just with a few steps, the Simulink model used for the simulation can be transformed into a real-time controller running on a standard PC.

## References

- Balemi, S., Moerschell, J., Breguet, J.-M., Brändlin, D., Bottinelli, S., & Beltrami, I. (2004). Surface inspection system for industrial applications. In *Conference on robotics and mechatronics*, Aachen, Germany (pp. 1597–1602).
- Beal, D., Bianchi, E., Dozio, L., Hughes, S., Mantegazza, P., & Papacharalambous, S. (2000). RTAI: real time applications interface. *Linux Journal*.
- Bianchi, E., & Dozio, L. (2000). Some experience in fast hard real-time control in user space with `rtai-lxrt`. In *Real time Linux workshop*, Orlando.
- Bianchi, E., Dozio, L., Mantegazza, P., & Ghiringhelli, G. L. (1999a). Complex control system, application of `diapm-rtai` at `diapm`. In *Real time Linux workshop*, Vienna.
- Bianchi, E., Dozio, L., Martini, D., & Mantegazza, P. (1999b). Applications of a hard real-time support in digital control of complex aerospace systems. In *AIDAA congress*, Torino, Italy.
- Bucher, R., & Dozio, L. (2003). CACSD with linux RTAI and RTAI-lab. In *Real time Linux workshop*, Valencia.
- Dozio, L., & Mantegazza, P. (2003a). Linux real time application interface (`rtai`) in low cost high performance motion control. In *Motion Control 2003*. Milano, Italy: ANIPLA.
- Dozio, L., & Mantegazza, P. (2003b). Real time distributed control systems using `rtai`. In *Sixth IEEE international symposium on object-oriented real-time distributed computing*, Hokkaido, Japan.