

# RAPID CONTROLLER PROTOTYPING WITH MATLAB/SIMULINK AND LINUX

Roberto Bucher \* Silvano Balemi \*

\* *University of Applied Sciences of Southern Switzerland (SUPSI), Department of computer science and electronics, CH-6928 Lugano-Manno, bucher@die.supsi.ch*

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 produced code runs as a kernel module on a standard PC with the modification of the Linux Operating System.

This environment can be used in a teaching laboratory to quickly implement real-time controllers. Students have the possibility to follow all the phases of the controller design within a unique environment (analysis, design, simulation, implementation) and can concentrate on the design aspects without having to deal with programming issues.

Some applications are presented to demonstrate the capabilities and the performance of this environment.

Keywords: Control applications, Education, Rapid Controller Prototyping

## 1. INTRODUCTION

Rapid prototyping can be successfully used in a control laboratory but usually at the cost of expensive software and hardware packages. These costs can be reduced by using free Linux software on a single standard PC and the commercial Matlab suite. Then, students can work within the same environment from the analysis to the controller implementation phase (see figure 1).

The considered “hardware in the loop” is a standard PC in which a common Linux OS is modified with the RTAI extension from the Dipartimento di Ingegneria Aerospaziale of the Politecnico di Milan (DIAPM). 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 microseconds.

The RTAI extension was created as an environment for implementing low cost data acquisition

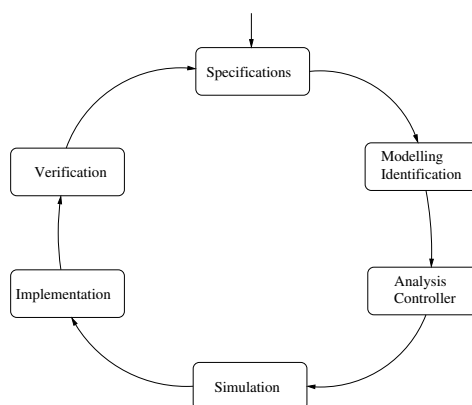


Fig. 1. Design phases

and digital controller systems ([D. Beal and Papacharalambous (April 2000)]). It has already reached a good level of maturity and has been widely used in different applications ([Bianchi and Dozio (2000)], [E. Bianchi and Ghiringhelli (1999)], [E. Bianchi and Mantegazza (1999)]). RTAI allows

to implement processes both in the kernel and in the user area.

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

Starting from the C-code generated by the Real-Time Workshop toolbox, the subsequent generation of executable code for processes in the user area of the extended Linux OS has been demonstrated ([Quaranta and Mantegazza (2001)]). The present work shows how to automatically generate code for the kernel area instead. This guarantees that processes run with highest priority, without being influenced by various other tasks running on the PC.

Such an integrated system is very valuable for control system laboratories. Students can design and implement controllers without having to explicitly program hardware. Moreover, the control hardware is given by standard PCs, thus limiting the investment needs and allowing later inexpensive upgrades of the control system hardware with newer, faster computers when higher performances are needed.

## 2. THE ENVIRONMENT

### 2.1 Hardware and software requirements

The system can be installed on a PC with following features:

- Linux with kernel 2.4.x modified with the RTAI extensions,
- Matlab, Simulink and Real-Time Workshop,
- C source code for the generation of the kernel modules, and the respective Matlab TLC (target language compiler) files and Template Makefiles,
- Acquisition and interface boards (analog, digital, encoders, ...),
- Simulink library with the drivers of the I/O boards.

### 2.2 RTAI extensions

The RTAI extensions can be downloaded from the web ([[www.aero.polimi.it](http://www.aero.polimi.it)]). The Linux OS can be modified to a hard real-time OS following these steps:

- The kernel has to be modified with the RTAI extension files, re-compiled and installed.
- The RTAI specific modules have to be compiled and installed.
- The RTAI devices (FIFOs, shared memories) have to be created.

These activities require less than one hour effort.

### 2.3 Matlab, Simulink and Real-Time Workshop

The Matlab/Simulink tool is very useful when teaching students how to analyze and design control systems. In spite of its cost, RTW is one of the best solutions to automatically generate C-code, and can be quickly adjusted to work with different targets. Mathworks Inc. allows Universities to acquire this software at special conditions.

### 2.4 The “rtwrtai” toolbox

This software represents the main part of our work. The toolbox can be freely downloaded at [www.die.supsi.ch/~bucher](http://www.die.supsi.ch/~bucher). A set of modules allows the integration of the code generated by Matlab, Simulink and RTW with the RTAI Linux environment. The most important modules are:

- The main module `rt_proc.c` which integrates the kernel specific functions and the Interrupt Service Routine.
- An intermediate module which combines the kernel specific task of the main module with the non-kernel code generated by RTW.
- The source code for specific Simulink modules (I/O drivers, scope) implemented as C-mex S-functions.
- The source code to access the I/O boards and the scope from the kernel module.
- The source code to implement specific functions not present at the kernel level (malloc, strcmp, memcpy, etc.).

### 2.5 Matlab TLC and Template Makefile

RTW requires two special files to allow the code generation for a specific target:

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

As a starting point for the creation of the TLC files we could use the default Target Language Compiler file with little modifications. The main problem was to obtain a Template Makefile which

was able to link kernel specific code (RTAI) with user specific code (RTW). This was only possible with the use of compilation flags handling the compilation of the different types of modules. The resulting Template Makefile allows also the use of several Matlab toolboxes (Signal Processing Toolbox, Fuzzy Control Toolbox, etc.).

The resulting files support both continuous and discrete-time blocks, which remarkably increases the potentiality of the tool.

### 2.6 I/O Boards

The use of acquisition and interface boards is essential when controlling real plants. For each function provided by a board (AD-conversion, DA-conversion, ...), two modules have to be programmed:

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

The C-mex S-function only handles 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 a A/D conversion and reading the resulting values.
- controlling a digital I/O.
- reading the counter of an incremental encoder.

### 2.7 RTAI library for Simulink

Figure 2 shows the Simulink library currently available.

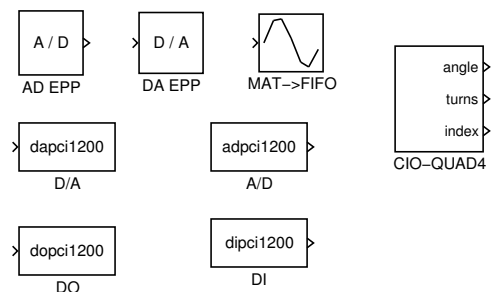


Fig. 2. RTAI specific Simulink Library

At present, the project supports the following boards from Computerboards:

- PCI-DAS1200 with 16 330KHz 12-Bit A/D converters with 3 $\mu$ s Burst Mode & Prog Gain and 2 D/A converters.

- CIO-QUAD04, 4 Channel quadrature encoder board.

The block “MAT->FIFO” allows to exchange data from the kernel area to the user area.

## 3. SOME EXAMPLES

### 3.1 Example 1

Figure 3 shows a simple Simulink model with two sinusoidal input signals.

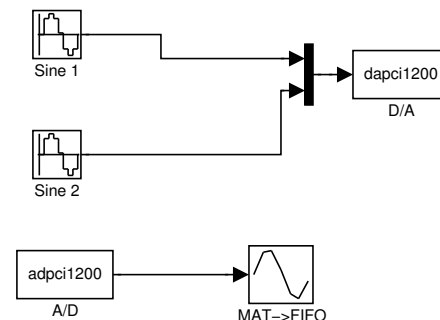


Fig. 3. Example 1 - Simulink Block Diagram with sinusoidal input signals

The signals are sent to the DAQ board and converted to analog voltages. The two voltages are read by the A/D converters of the same board and sent to the scope. The scope saves the values into a FIFO, which allows to retrieve the values in the user area. The signals are eventually plotted using “gnuplot” (see Figure 4).

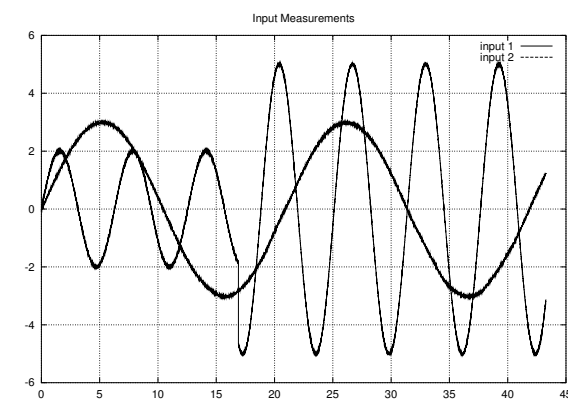


Fig. 4. Example 1 - Data plot

Parameters can be modified online during the execution of the real-time code. This feature has been programmed using the “Parameters Tuning” capability implemented in RTW. A real “External Mode” has not been implemented yet.

### 3.2 Example 2

This example, similar to the previous one, shows the possibility to implement blocks with multiple

sampling rates. The first signal is sampled at 1KHz, the second at 5Hz. Figure 5 shows the data plotted after the acquisition.

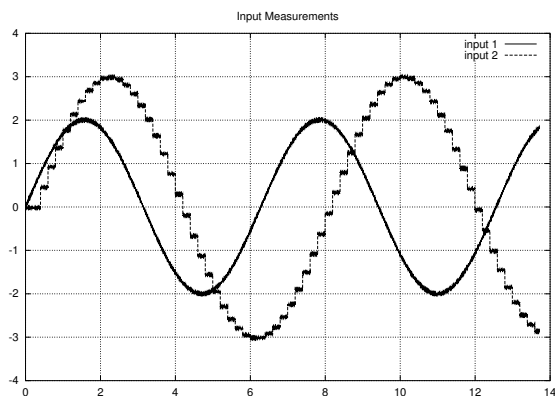


Fig. 5. Example 2 - Data plot

### 3.3 Example 3

This example shows how students can control an inverted pendulum. A state-feedback controller is used, the unmeasured states of the plant are estimated using a reduced order observer. Figure 6 shows the Simulink model used by the students to simulate the controlled plant.

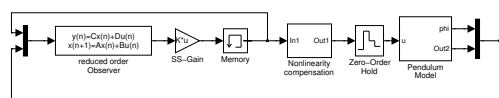


Fig. 6. Example 3 - Simulation model for the inverted pendulum

Figure 7 shows the Simulink model used to generate the real-time control program: there are very few modifications with respect to the simulation model shown in Figure 6.

The model of the plant is simply substituted by the I/O modules and the controller is now ready for implementation. Writing by hand the code for this controller can be very time-consuming. Automatic code generation in this case, can remarkably reduce the implementation time. Students can concentrate on variants of the controller without spending a lot of time in programming.

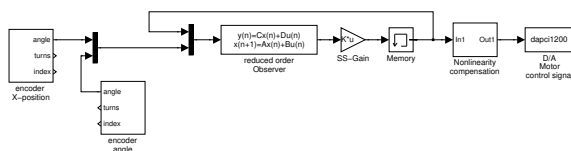


Fig. 7. Example 3 - Controller implementation

## 4. STUDENT FEEDBACK

Student feedback is very positive. Above all, students appreciate the different possibilities to design, implement and test a controller within the

same environment. Furthermore they can start implementing controllers very early, even without knowledge of discrete-time systems: continuous and discrete-time blocks can be mixed in the same Simulink model.

### 4.1 Continuous-time controllers

The system allows the implementation of continuous-time transfer functions. An integration algorithm is automatically integrated into the generated code if continuous blocks are defined. As soon as the students have the first knowledge in analysis and design of classical controllers, they can implement the first controllers and test how they work with real plants.

### 4.2 Identification

Possibility is given to collect I/O data for identification. Different input signals (step, random, PBRs, ...) can be applied to the plant in order to collect input and output signal sets. Afterwards these data can be analyzed offline in order to identify the plant. Figure 8 shows a simple block diagram used to collect data for identification.

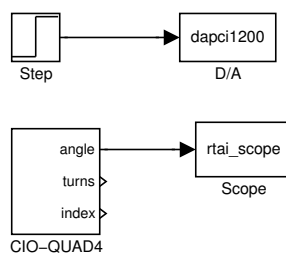


Fig. 8. A simple schema to collect data

### 4.3 Controller algorithms

One of the drawbacks of the use of rapid controller prototyping systems in teaching activities is that students are not faced to the issues raised by writing and implementing controller algorithms by hand. However, this system allows to simply extend generated code with code parts written by hand. The code generated by RTW can be modified in order to call external procedures.

For example a discrete-time PID controller written in C by hand in a file can be simply merged into the RTW code thus avoiding tedious and inefficient work (e.g. writing or integrating code for card drivers). The code corresponding to the block diagram of figure 9 given by the following lines

```
rtb_temp9 = (rtB.Pulse_Generator -
             rtB.S_Function_o1) * rtP.controller_Gain;
```

can be modified in order to call the external PID controller code as follows.

```
err = (rtB.Pulse_Generator - rtB.S_Function_01);
rtb_temp9 = reg_pid(err);
```

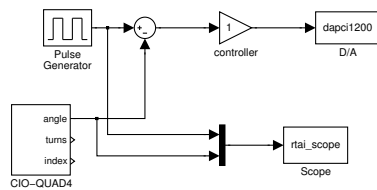


Fig. 9. Schema to implement self written code

#### 4.4 Student involvement in the project

This project began with a student project focused on a feasibility study. Now students participate in the further development of the environment by implementing new drivers during laboratory activities and student projects.

## 5. CONCLUSIONS

A simple and inexpensive tool for use in control system laboratories has been presented. This tool allows to reach sampling frequencies up to approximately 10KHz, which is more than enough for controlling most plants in student laboratories.

Using the “rtwrtai” toolbox, the Matlab/Simulink environment can be used for acquiring data to identify the plant, for analysis and for 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.

The present work provides a valid tool for rapid controller prototyping systems while minimizing the cost of the control hardware.

## REFERENCES

- E. Bianchi and L. Dozio. Some experience in fast hard real-time control in user space with rtaixrt. In *Real Time Linux Workshop*, Orlando, 2000.
- L. Dozio S. Hughes P. Mantegazza D. Beal, E. Bianchi and S. Papacharalambous. Rtai: real time applications interface. *Linux Journal*, April 2000.
- D. Martini E. Bianchi, L. Dozio and P. Mantegazza. Applications of a hard real-time support in digital control of complex aerospace systems. In *AIDAA Congress*, Torino, Italy, 1999.
- P. Mantegazza E. Bianchi, L. Dozio and G. L. Ghiringhelli. Complex control system, application of diamp-rtai at diamp. In *Real Time Linux Workshop*, Vienna, 1999.

G. Quaranta and P. Mantegazza. Using matlab-simulink rtw to build real time control application in user space with rtai-lxrt. In *Real Time Linux Workshop*, Milano, 2001.  
www.aero.polimi.it.