USING SCILAB/SCICOS WITH RTAI-LAB

Roberto Bucher
University of Applied Sciences of Southern Switzerland
Galleria 2, Manno 6928, C H
roberto.bucher@supsi.ch

1 Introduction

Computer Aided Control System Design (CACSD) subsumes a broad variety of computational tools and environments for control system design, real time simulation, with and without hardware in the loop, making the best use of high desktop computer power, graphical capabilities and ease of interaction with low hardware cost. Integrated CACSD software environments allow an iteractive control system design process to be automated with respect to multi-objective performances evaluation and multi-parameter synthesis tuning. Visual decision support provides the engineer with the clues for interactively directing an automated search process to achieve a well balanced design under many conflicting objectives and constraints. Local/remote on line data down/upload makes it possible a seamless interaction with the control system, in order to supervise its operation and to adapt it to changing operational needs.

2 Installing the RTAI add-ons for Scilab/Scicos

2.1 Installing RTAI-Lab

2.1.1 Getting the files and the libraries

First of all you need RTAI-Lab. RTAI-Lab is an open project integrated in the Linux RTAI distribution. In order to run RTAI-Lab you need the following packages:

Linux RTAI download the last stable release from the RTAI homepage www.rtai.org

Mesa libraries download this libraries using the following steps:

1. Download MesaLib-6.2.tar.gz (from www.mesa3d.org) in a temporary directory (/tmp)
2. Untar the archive : tar xvzf MesaLib-5.0.1.tar.gz
3. cd /tmp/Mesa-6.2
4. make linux-86
5. make install
   - /usr/include + /usr/lib
6. make install (
   - /usr/X11R6/include + /usr/X11R6/lib

Compile and install the EFLTK package

1. Download EFLTK from CVS in a temporary directory (/tmp)
   cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ede login
   (press ENTER when CVS asks for password)
   cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ede co efltk
2. cd /tmp/efltk
3. ./build.gcc –prefix=/usr/local –enable-xdbe –enable-opengl –enable-threads
4. make install
5. /sbin/ldconfig

2.1.2 Installing RTAI and RTAI-Lab

Follow these steps in order to installa Linux RTAI

- Kernel
    1. Get a "vanilla" kernel from www.kernel.org
    2. Unpack the kernel
    3. Unpack Linux RTAI
    4. Patch the kernel with the adeos patch (patch -p1 ¡ ¡rtaidir¿/rtai-core/arch/i386/patches/¡kernel-version¿.patch
    5. make xconfig
    6. configure the kernel
    7. make bzImage
    8. make modules
    9. make modules_install
    10. make install
    11. fit lilo or grub for this new kernel
- install COMEDI : install comedi and comedilib from www.comedi.org
- install RTAI
    1. Go to the RTAI directory
    2. make xconfig
    3. Check "COMEDI support under LXRT" in the Add-ons submenu and give the directory where you installed comedilib
    4. Check "RTAI-Lab" under the RTAI-Lab submenu and add the directory where you installed the efltk libraries (normally "/usr/local")
    5. save and exit from xconfig
    6. run "make" and "make install"
- IMPORTANT: Add "/usr/realtime/bin" in your PATH environment variable, by modify the "/etc/profile" file or the ".bashrc" in the user direcory

2.1.3 Installing Scilab

The first step is to download and install the full source Scilab-3.0 release from "www.scilab.org"; it contains Scicos already. Install scicos:

1. Go in the directory "/usr/local"
2. Unpack Scilab
3. cd scilab-3.0
4. ./configure
5. make all

2.1.4 Installing the RTAI add-ons for Scilab

Now follow these steps to properly install all the Scilab/Scicos add-ons for RTAI-Lab:

1. become superuser
2. go in the "macros" directory found here
3. modify eventually in the file "Makefile" the line

   ```
   SCILAB_DIR = /usr/local/scilab-3.0
   ```

   to fit your SCILAB installation
4. run "make install"

Each user who want to work with the Scilab/Scicos RTAI add-ons has to modify his own ".scilab" file. This operation can be done running as normal user "make user" in this "macros" directory. This command add the following lines to the user ".scilab" file:

```
load('SCI/macros/RTAI/lib')
%scicos_menu($+1)=['RTAI','RTAI CodeGen']
scicos_pal($+1,:)=['RTAI-Lib','SCI/macros/RTAI/RTAI-Lib.cosf']
```

These lines add the menu "RTAI→CodeGen" to the scicos window and the new RTAI-Lib.cosf library with the RTAI specific blocks to the scicos palette.
At this point you are ready to generate code for RTAI, using the new menu "RTAI→RTAICodeGen" in the scicos window.

3 A simple example

3.1 Scheme

In the following, a simple example will be analized. The system is represented by a transfer function

$$Gs(s) = \frac{20}{s^2 + 4s}$$

with unity feedback,
The system has been implemented as discrete-time transfer function

$$Gz(z) = 10^{-6} \frac{9.987z + 9.973}{z^2 - 1.996z + 0.996}$$

with a sampling time of $1ms$. Different signals are sent to scopes, meters, and leds.
The model is saved with the name "test".

3.2 Implementation under Scilab

3.2.1 Designing the scheme

First of all we have to design the system using scicos. We can get the different blocks from the scicos palettes in order to obtain the desired system. By the next step we have to integrate some I/O into our scheme. We implemented three methods:

- I/O can be choosed from a specific RTAI Library
- I/O were configured by hand

These methods can be mixed together.

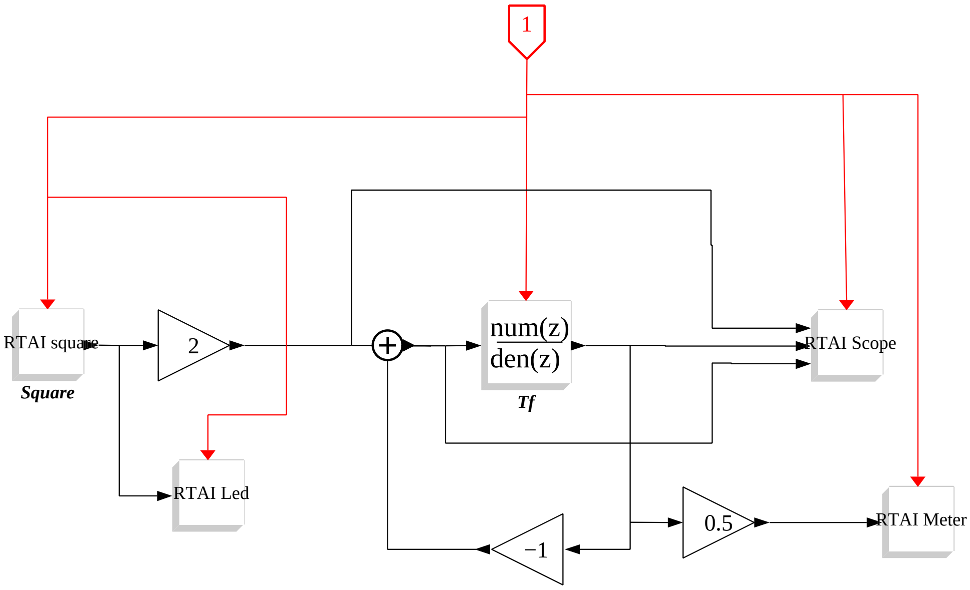

Figure 1: Scicos scheme

3.2.2 Implementing I/O using the RTAI palette

Figure 1 represents the Scicos scheme of the example in this case.
It is important to give to each I/O input and output a different Port nr:

- RTAI_scope: 1
- RTAI_meter: 2
- RTAI_led: 3
- RTAI_square: 1

In order to generate the code this scheme must be transformed into a "Super Block" which can be used to generate the code. The menu "Diagramm" "Region to Super Block" allows to select a part of the scheme and to put it into a "Super Block". We can access to the Super Block scheme simply by clicking on it. A good idea is to open the "Super Block" and to rename it ("Diagram" "Rename") to "test". This will be the name of the generated model and of the directory where the generated files are stored.
Now we can simply choose the menu "RTAI" "RTAICodeGen" to generate and compile the realtime code.
A dialog Box asks about some compilation parameters: the most important are

- the sampling time (the proposed value have been read from the clock block connected to the superblock).
- the name of the configuration file (default is "config").

After "OK" scicos performs the code generation and the compilation of the generated modules.

4 I/O Blocks

4.1 Basics

I/O blocks are implemented in a library (libsciblk.a). There are 3 kind of I/O blocks:

1. Blocks which can be connected to an input or an output port (ex. rtai_comedi_data)

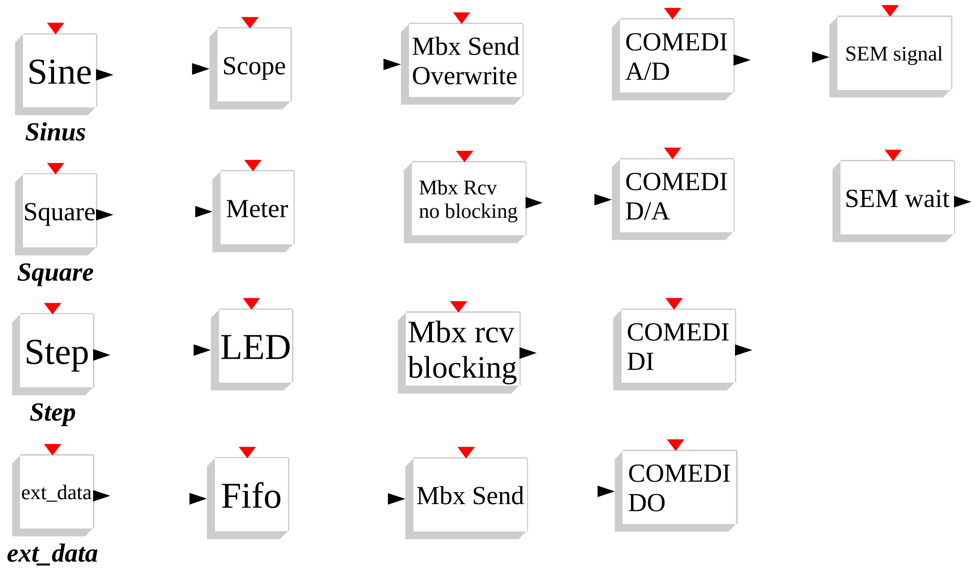

Figure 2: Scicos palette for RTAI

2. Blocks which can be connected only to an input port (ex. step)
3. Blocks which can be connectes only to an output port (ex. rtai_scope)

4.2 Available blocks

Figure 2 shows the palette with the present available RTAI blocks under Scilab/Scios.

5 Implementation of new user blocks

5.1 Implementing the code for a IO device

Different tools facilitate the implementation of new blocks. In order to implement new drivers some skeletons are provided:

- "template.c"
- "devtmpl.h"

Using the command

```
gen_dev <model>
```

a file "<model>.c" will be created. This file contains all the needed functions to implement the driver as "input" and "output" driver. The utility "gen_dev" fills the file "devices.h" too.
The file "devstruct.h" contains the description of the structure used to store the block specific datas.

```
typedef struct devStr{
  int nch;
  char IOName[20];
  char sName[20];
  char sParam[20];
  double dParam[5];
  int i1;
  long l1;
```

```
  long l2;
  void * ptr;
}devStr;
```

A structure for input (inpDevStr) and a structure for output (outDevStr) are provided in "rtmain.c". Basically, the channel information can be stored under the field "nch", the name under "sName" and the 5 parameters p1...p5 into the field "dParam". The field IOName is used to store the name of the IO Block needed by the online parameter modification.
The function interfaces in the generated file reflect the call implemented in <model_io.c> after calling "gen_io". Now the user can simply implement the code for the IO block.

5.2 Create the new library file

In order to generate the new library file with the new implemented IO Block, the user must modify the "Makefile.am" file (or "GNUmakefile.am" in the last RTAI releases), adding <model>.c to the list of the files to be compiled (libsciblk_a_SOURCES). The user have to launch "automake rtai-lab/scilab/devices/Makefile" (or "automake rtai-lab/scilab/devices/GNUmakefile") from the RTAI root directory, followed by "./config.status". Now he can run "make" and "make install' in the scilab/devices directory. The libsciblk.a file will be created and copied in the library directory.

5.3 Creating the Scicos block

To be done ...

6 Using COMEDI drivers