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

Abstract

This paper describes how to implement a controller for the Rtai-Lab environment using Scilab/Scicos.

In the second part of this paper the generation of user specific IO blocks is presented.

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

After installing Scilab-2.7 the RTAI patches can be applied. Simply run "make install" in the directory "RTAI_ROOT¿/rtai-lab/scilab/macros. This command performs the following steps:

1. Add the macro "RTAICodeGen_.sci" to the directory "<SCIDIR>/macros/scicos".
2. Add "RTAICodeGen_.sci" to the "Makefile" in the directory "<SCIDIR>/macros/scicos".
3. Modify the macro "initial_scicos_tables.sci" under "<SCIDIR>/macros/util" by adding the menu "RTAI" to the scicos window.
4. Compiles these new macros.

5. Modify some scicos palettes in order to identify some blocks with a name for parameter transfer.

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

First of all we have to design the system using scicos. Figure **??** represents the Scicos scheme of the example.

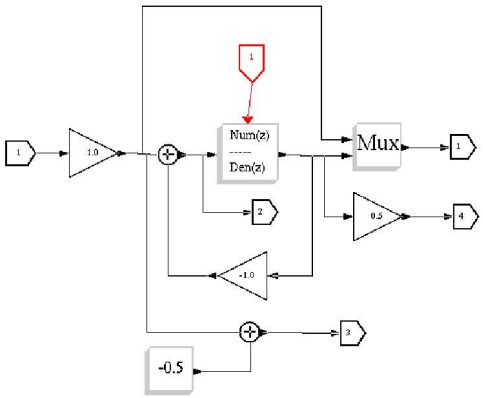

FIGURE 1: *Scicos scheme*

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 (default is 1ms).
- the name of the configuration file (default is"confi").

After "OK" scicos performs the code generation and asks about the creation of the IO configuration file. There are 3 possible choices here:

Yes scicos open "xgenconfig" to create or modify the IO configuration file, generates the <model_io>.c file and generates the standalone executable.

No scicos try to use an already created IO configuration file to genearate the <model_io>.c file and then performs the compilation of the standalone executable.

Cancel scicos stops the code generation.

All this operations can be performed in a linux shell. Two utilities are provided to generates IO configuration files:

genconfig a simple text system to create the "config" file

xgenconfig a graphical environment to create IO configuration files. Simply run "xgenconfig -iN -oM -f<config_name> to create the <config_name> configuration file with N input and M output.

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)
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)

For each block in the config file the following (block dependent) items have been defined:

- the type (example: rtai_scope)
- input (inp) or output (out) port (where the block is connected)
- the port number to which the block is connected
- an identifier, a channel number or a maximal number of channels/signals for this block
- a name
- 5 parameters

4.2 Available blocks

The following IO Blocks have been implemented:

sinus creates a sinus input signal

square creates a square inpit signal

step creates a step input signal

rtai_scope sends signal(s) to the Rtai-Lab scope widget

rtai_led sends signal(s) to the Rtai-Lab leds widget

rtai_meter sends signal to the Rtai-Lab meter widget

mem allows to connect an output port to an input port of the scicos modul

rtai_comedi_data allows to connect COMEDI drivers for analog signals to the scicos modul (input or output)

rtai_comedi_dio allows to connect COMEDI drivers for digital signals to the scicos modul (input or output)

extdata gets the data from a file and creates a periodic input signal

pcan allows to connect the scicos modul to a Maxon driver through a bus can using a Peaks epp-dongle

cioquad4 build an input signal getting data from a Computer Boards CIO-QUAD4 digital encoder card

mbx_receive_if asynchronous receiver from a named mailbox (blocking) for distributed systems

mbx_receive receiver from a named mailbox (not blocking) for distributed systems

mbx_ovrwr_send sender to a named mailbox for distributed systems

The tables 1 . . . 17 show the parameters description of the IO Block.

io	inp
port	port Nr.
ch	not used
name	not used
sParam	not used yet
p1	Amplitude
p2	Frequency
p3	Phase
p4	Bias
p5	Delay

Table 1: Parameters for the IO block: ***sinus***

io	inp
port	port Nr.
ch	not used
name	not used
sParam	not used yet
p1	Amplitude
p2	Period
p3	Impulse width
p4	Bias
p5	Delay

Table 2: Parameters for the IO block: ***square***

io	inp
port	port Nr.
ch	not used
name	not used
sParam	not used yet
p1	Amplitude
p2	Delay
p3	not used
p4	not used
p5	not used

Table 3: Parameters for the IO block: ***step***

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];
```

io	out
port	port Nr.
ch	number of signals
name	Scope name
sParam	not used yet
p1	not used
p2	not used
p3	not used
p4	not used
p5	not used

Table 4: Parameters for the IO block: ***rtai_scope***

io	out
port	port Nr.
ch	number of leds (1 . . . 16)
name	Led name
sParam	not used yet
p1	not used
p2	not used
p3	not used
p4	not used
p5	not used

Table 5: Parameters for the IO block: ***rtai_led***

```
  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, 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" from the RTAI root directory, followed by "./config.status". Now he can run "make" and "make install' in the

io	out
port	port Nr.
ch	not used
name	Meter name
sParam	not used yet
p1	not used
p2	not used
p3	not used
p4	not used
p5	not used

Table 6: Parameters for the IO block: ***rtai_meter***

io	inp or out
port	port Nr.
ch	id (0,1,2,...)
name	not used
sParam	not used yet
p1	not used
p2	not used
p3	not used
p4	not used
p5	not used

Table 7: Parameters for the IO block: ***mem***

scilab/devices directory. The libsciblk.a file will be created and copied in the library directory.

5.3 Adapting the "genconfig" and "xgenconfig" utility

The "genconfig" and "xgenconfig" utilities must now be modified to include the question for the new device parameters. This can be done by modifying the file "config_data.h", which contains two matrices of strings with the different questions. Simply increment the number of blocks (input or output or both) changing the values of the two "#define", and add a line to the repectives matrix providing the following strings:

1. the name of the block
2. the question to the "nch" parameter
3. the question to the "sName" parameter
4. the question to the "sParam" parameter
5. the questions to the "p1"..."p5" parameters

Now we can simply run "make" and "make install' to generate and install the new "genconfig" and "xgenconfig" files.

io	inp or out
port	port Nr.
ch	ch or number of signals (device dependent)
name	"comediX"
sParam	not used yet
p1	range (device dependent)
p2	aref (device dependent)
p3	not used
p4	not used
p5	not used

Table 8: Parameters for the IO block: ***rtai_comedi_data***

io	inp
port	port Nr.
ch	ch or number of signals (device dependent)
name	"comediX"
sParam	not used yet
p1	not used
p2	not used
p3	not used
p4	not used
p5	not used

Table 9: Parameters for the IO block: ***rtai_comedi_dio***

io	out
port	port Nr.
ch	ch or number of signals (device dependent)
name	"comediX"
sParam	not used yet
p1	threshold
p2	not used
p3	not used
p4	not used
p5	not used

Table 10: Parameters for the IO block: ***rtai_comedi_dio***

io	inp
port	port Nr.
ch	number of values
name	file name
sParam	not used yet
p1	not used
p2	not used
p3	not used
p4	not used
p5	not used

Table 11: Parameters for the IO block: ***extdata***

io	inp
port	port Nr.
ch	not used
name	CAN id (hex)
sParam	not used yet
p1	Proportional driver gain (PI)
p2	Integral driver gain (PI)
p3	not used
p4	not used
p5	not used

Table 12: Parameters for the IO block: ***pcan***

io	out
port	port Nr.
ch	not used
name	CAN id (hex)
sParam	not used yet
p1	Proportional driver gain (PI)
p2	Integral driver gain (PI)
p3	not used
p4	not used
p5	not used

Table 13: Parameters for the IO block: ***pcan***

io	inp
port	port Nr.
ch	modul number
name	Base Card Address
sParam	not used yet
p1	Resolution
p2	Precision (1,2,4)
p3	Rotation (-1,+1)
p4	Initial (0) or continous reset (1)
p5	not used

Table 14: Parameters for the IO block: ***cioquad4***

io	inp
port	port Nr.
ch	number of signals
name	IP Address
sParam	MBX Name
p1	not used
p2	not used
p3	not used
p4	not used
p5	not used

Table 15: Parameters for the IO block: ***mbx_receive_if***

io	inp
port	port Nr.
ch	number of signals
name	IP Address
sParam	MBX Name
p1	not used
p2	not used
p3	not used
p4	not used
p5	not used

Table 16: Parameters for the IO block: ***mbx_receive***

io	out
port	port Nr.
ch	number of signals
name	IP Address
sParam	MBX Name
p1	not used
p2	not used
p3	not used
p4	not used
p5	not used

Table 17: Parameters for the IO block: ***mbx_ovrwr_send***