RTAI-Lab tutorial: Scilab, Comedi, and real-time control

Roberto Bucher Simone Mannori Thomas Netter

Version 0.50, 4th November 2005

1 Introduction

Computer Aided Control System Design (CACSD) includes a wide range 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 interactive 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 adapt it to changing operational needs.

We present a full development chain which uses software tools released under open-source licenses. Most of them are released under GNU/GPL. Scilab/Scicos have a license which is very similar, but not exactly equal, to GNU/GPL. All the software used here is Patent-Free. In this way the cost of the software acquisition is reduced around zero and there are not recurring costs or constraints like royalty or patents. The end-user is free to use commercial I/O hardware devices (supported by ”ready to use” Comedi device drivers) - or any other kind of hardware - also custom developed - writing a Comedi driver or direct control code embedded in SCICOS blocks. These aspects are real strategic advantages - compared with a proprietary solution - both for educational and industrial applications.

2 Installation

To implement and use a complete CACSD developement chain you need:

- a real time operating system (Linux kernel with ADEOS patch and the RTAI library);
- a design and developement interactive graphical environnement (SCILAB/SCICOS);
- an automatic code generator: from visual symbolic representation to real code to compile (integrated in RTAI);
- a user interface to display the behaviour of the system and interact with (RTAI-Lab is included in RTAI project);
- a library of I/O driver (Comedi and Comedilib, kernel and user space driver and libraries);
- the possibility to encapsulate direct access code to I/O devices and/or RTAI functions inside a SCICOS block (we are giving here some simple examples).

2.1 Be politically incorrect

In a Linux system, only the user ”root” has the privilege to do most of the operations described in this document. Instead of login as normal user then switch to ”superuser” (”root”) when it is necessary, we suggest you to work as ”root”. It is a little bit dangerous, because as ”root” you can clear all the system files. However, it is faster than typing ”su” plus ”password” many times. Make your choice. Think before press [Enter] key.

2.2 Getting libraries

A good starting point is a complete Linux distribution as Debian, Fedora, Slackware, Ubuntu, Suse, etc... with the default kernel developement tools (GCC) in a graphical environnement (Gnome or KDE).
We strongly suggest a full distro installation to be sure to have all the developement tools and libraries.

2.2.1 Installing Mesa and EFLTK libraries

As prerequisite you need to download, configure compile and install Mesa and EFLTK.

EFLTK is a library of graphical primitive used by RTAI-Lab user interface. EFLTK is light and fast because all its functions use OpenGL calls. These calls are handled by Mesa and, if you have a 3D accelerated videocard supported in the kernel and RTAI compatible, all the graphic user interface (RTAI-Lab) is rendered in hardware with minimal CPU load using DRI (Direct Rendering Interface). If the DRI support is not available (for hardware or maximum compatibility reasons) all the OpenGL primitives are handled in software emulation by the Mesa library using some CPU effort.

The 3D-DRI hardware support in the kernel is ADEOS-RTAI tested and garantee only with kernel 2.4.x. We have tested, with positive results, Intel integrated i8xx, ATI (up to Ati 9250) and Matrox. The 2.4.x. DRI support configuration is described in appendix.

For reasons which go beyond our knowledge, the DRI support inside in the 2.6.x kernel is not compatible with ADEOS-RTAI. With 2.6.x we had positive results only using the latest proprietary ATI driver. To use this driver you need, obiouvsly, a ATI videocard (see www.ati.com for the list of supported cards), and you have to follow a specific procedure (reported in this document) for the kernel, external modules and X11 driver configuration, compilation and installation.

The proprietary nVidia driver (www.nvidia.com) is very performing and easy to install, but it is not RTAI compatible because it blocks the interrupts for too much time and, as a consequence, it freezes the machine if a realtime task is active.

The video performance is critical if you need complex graphics visualization (e.g. realtime user interface and multimedia).

Mesa libraries download and install these libraries using the following steps:

1. Warning: In some distributions the Mesa 3D support is pre-installed. In any case, don’t ”jump” the Mesa installation. You really need this package installed from the source to obtain good results. If you jump this step the results will be incomplete rendering of the RTAI-Lab graphics.
2. Download MesaLib-6.2.tar.gz (from www.mesa3d.org) in a temporary directory (/tmp)
3. Untar the archive : ”tar xvzf MesaLib-6.2.tar.gz”
4. ”cd /tmp/Mesa-6.2”
5. ”make linux-86” or ”make linux-x86-static”. Both are working solutions.
6. ”make install”. This program makes a couple of questions regarding ”where” putting the library. Normally the suggested default are correct: answer [Enter] to both questions. The defaults are:

- ”/usr/include and” ”/usr/lib”
  But in some systems you may need to put the Mesa library files in different places, such as
- ”/usr/X11R6/include” and ”/usr/X11R6/lib”

If you are interested in absolute maximum performance, you need to compile the DRI support statically in the Mesa using the instruction in this DRI Wiky page (http://dri.freedesktop.org/wiki/).

EFLTK libraries 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 ”
   ”./efltk-config.in −−prefix=/usr/local −−multithread”
   ”./emake ”
   ”./emake install ”
3. Using an editor (e.g. gedit) add the path [/usr/local/lib] in the dynamic library configuration file [/etc/ld.so.conf].
   gedit /etc/ld.so.conf. The file should be like this one

   ```
   /usr/local/lib
   include ld.so.conf.d/*.conf
   /usr/X11R6/lib
   /usr/lib/mysql
   ```
4. Then run ”ldconfig” To update the library database(”ldconfig” or ”/sbin/ldconfig”)

2.3 Installing a patched kernel for RTAI and RTAI-Lab

- Kernel (case 2.4.x). The latest RTAI v.3.2 supports kernel 2.4.27 with ”hal-linux-2.4.27-i386-r15.patch”.
  1. Get a ”vanilla” kernel from www.kernel.org (e.g. ”linux-2.4.27.tar.bz2”)
  2. Get RTAI from www.rtai.org (e.g. ”rtai-3.2.tar.bz2”)
  3. Change working directory to /usr/src (”cd /usr/src”)
  4. Unpack the kernel (”tar xjvf linux-2.4.27.tar.bz2”)
  5. Unpack Linux RTAI (”tar xjvf rtai-3.2.tar.bz2”)
  6. Make linux symbolic link (”ln -s /usr/src/linux-2.4.27 linux”)
  7. Make rtai symbolic link (”ln -s /usr/src/rtai-3.2 rtai”). These links semplify next installation steps.
  8. Patch the kernel with the ADEOS patch
     generic ”patch -p1 < <rtaidir>/base/arch/i386/patches/<kernel-version>.patch” or
     specific ”patch -p1 < /usr/src/rtai/base/arch/i386/patches/hal-linux-2.4.27-i386-r15.patch”)
  9. ”make xconfig” (or ”make menuconfig” for text interface)
  10. configure the kernel
  11. ”make bzImage”
  12. ”make modules”
  13. ”make modules_install”
  14. ”make install”
  15. fit lilo or grub for this new kernel (see appendix).
- Kernel (case 2.6.x). The latest RTAI v.3.2 support kernel 2.6.10 with ”hal-linux-2.6.10-i386-r9.patch”

1. Get a ”vanilla” kernel 2.6.x from www.kernel.org (e.g. ”linux-2.6.10.tar.bz2”; anyway check first the availability of the corresponding ADEOS patch);
2. Get RTAI from www.rtai.org (e.g. ”rtai-3.2.tar.bz2”)
3. Change working directory to /usr/src (”cd /usr/src ”)
4. Unpack the kernel (”tar xjvf linux-2.6.10.tar.bz2 ”)
5. Unpack Linux RTAI (”tar xjvf rtai-3.2.tar.bz2 ”)
6. Make linux symbolic link (”ln -s /usr/src/linux-2.6.10 linux”)
7. Make rtai symbolic link (”ln -s /usr/src/rtai-3.2 rtai”). These links semplify next installation steps.
8. Patch the kernel with the adeos patch
   generic ”patch -p1 $<$ $<$rtaidir$>$/base/arch/i386/patches/$<$kernel-version$>$.patch ” or
   specific ”patch -p1 $<$ /usr/src/rtai/base/arch/i386/patches/hal-linux-2.6.10-i386-r9.patch ”
9. ”make xconfig”
10. configure the kernel
11. ”make”
12. ”make modules_install”
13. ”make install”.
14. fit lilo or grub for this new kernel (see appendix).

2.4 Installing the COMEDI and Comedilib files

The Comedi project develops open-source drivers, tools, and libraries for data acquisition. Comedi is a collection of drivers (+100 devices are supported) for a variety of common data acquisition plug-in boards. The drivers are implemented as a core Linux kernel module providing common functionality and individual low-level driver modules. In this way it is possible to write modular program that needs minimal or NO modifications if the I/O hardware is changed. There are primitives to ”inspect” the I/O device and dinamically configuring its features.
We strongly suggest to install the software from CVS, install and check the ”comedilib/doc/comedilib.pdf” BEFORE buying a new data acquisition card. If you already have a card NOT supported by Comedi don’t PANIC: with some documentation it is possible to adapt a similar Comedi driver or write a specific code to address the card functions.

Comedilib is a user-space library that provides a developer-friendly interface to Comedi devices. Included in the Comedilib distribution there is documentation, configuration and calibration utilities, and demonstration programs.

Kcomedilib is a Linux kernel module (distributed with Comedi) that provides the same interface as Comedilib in kernel space, suitable for real-time (RTAI) tasks. It is effectively a ”kernel library” for using Comedi from real-time tasks.

Comedilib Install Comedilib with the following commands

1. `cd /usr/src`
2. ```
   cvs -d :pserver:anonymous@cvs.comedi.org:/var/cvs login
   cvs -d :pserver:anonymous@cvs.comedi.org:/var/cvs co comedi
   cvs -d :pserver:anonymous@cvs.comedi.org:/var/cvs co comedilib
   ```
3. `cd comedilib`
4. Read software installation requirements in `README.CVS` and verify that your packages (automake etc.) are up to date with `automake --version`
5. `sh autogen.sh`
   ```

6. `./configure --sysconfdir=/etc`
7. `make` and `make install`
8. `make dev`
   This step created the `/dev/comedi`[0-3] device inodes. See the Comedi manual and `man comedi_config` to associate a particular driver and hardware device to one of the `/dev/comedi` device files.
   WARNING: with the new UDEV platform there are some problems if the new inodes are not registered.

RTAI (1st pass) Install RTAI (without Comedi support)

1. `cd /usr/src/rtai`)
2. `make xconfig` or `make menuconfig`
3. Menu General: verify default directories:
   - Installation directory `/usr/realtime`
   - Kernel source directory `/usr/src/linux`
4. Menu Machine (x86): adjust Number of CPUs (default = 2)
5. Exit xconfig/menuconfig and save configuration
6. `make` and `make install`
7. **IMPORTANT:** Add `/usr/realtime/bin` to the PATH variable in /etc/profile or your home directory's `.bashrc`.

Comedi for Linux RTAI

1. `cd /usr/src/comedi`
2. `sh autogen.sh`
3. `./configure` or possibly:
   `./configure --with-linuxdir=/usr/src/linux-2.6.10 --with-rtaidir=/usr/realtime`
4. `make`
5. `make install` (installs the comedi kernel modules)
6. `cp include/linux/comedi.h /usr/include/`
   `cp include/linux/comedilib.h /usr/include/`
7. `ln -s /usr/include/comedi.h /usr/include/linux/comedi.h`
   `ln -s /usr/include/comedilib.h /usr/include/linux/comedilib.h`

RTAI (2nd pass) (with Comedi support)

1. `cd /usr/src/rtai`)
2. `make xconfig` or `make menuconfig`
3. Menu Add-ons:
   - Select COMEDI support over LXRT
   - Specify COMEDI installation directory (`/usr/local` or `/usr`). The directory should contain `lib/libcomedi.a`, `include/comedi.h` and `include/comedilib.h`
4. Menu RTAI Lab:
   - Select RTAI Lab
   - Adjust EFLTK installation directory (default is `/usr/local`)
5. Exit xconfig/menuconfig and save configuration
6. `make` and `make install`

2.5 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 (”su”)
2. ”cd /usr/src/rtai/rtai-lab/scilab/macros”
3. modify eventually in the file ”Makefile” the line

   ```
   SCILAB_DIR = /usr/local/scilab-3.1.1
   ```

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

Each user who wants to work with the Scilab/Scicos RTAI add-ons has to modify his own ”.scilab” (scilab-3.0) or ”.Scilab/scilab-3.1.1/.scilab” (scilab-3.1.1) startup file. This operation can be done running as normal user ”make user” in this ”macros” directory. This command add the following lines to the startup file:

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

These lines add the menu ”RTAI→CodeGen” and ”Set Target” to the scicos window and the new RTAI-Lib.cosf library with the RTAI specific blocks to the scicos palette.
In order to finish the installation of Scilab you have to create a link to the scilab application under ”/usr/local/bin”:

```
cd /usr/local/bin
ln -s /usr/local/Scilab-3.1.1/bin/scilab scilab
```

Or add ”/usr/local/scilab-3.1.1/bin” to the PATH.

————————- END OF INSTALLATION SECTION ————————————

3 A simple example

3.1 Continous and time-sampled models

In the following, a simple example will be analysed. 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”.

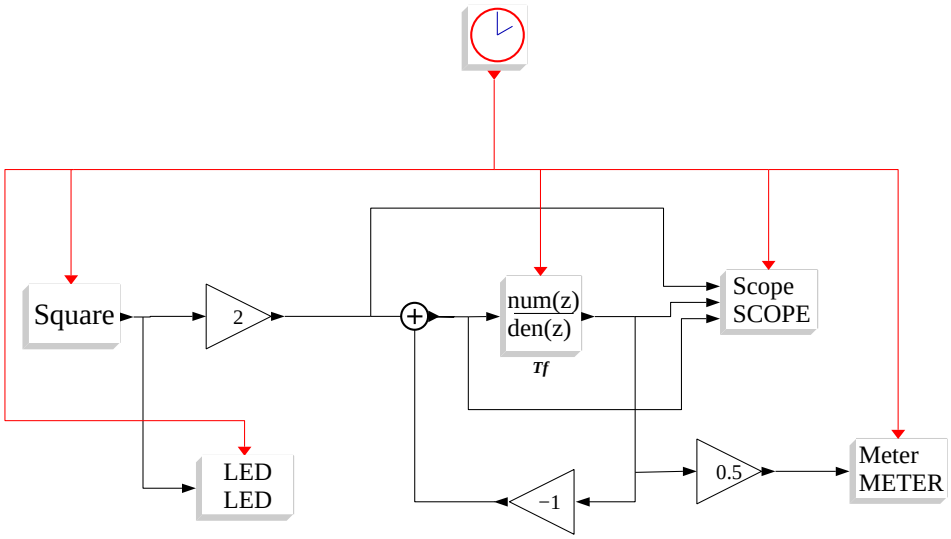

Figure 1: Scicos block diagram

3.2 Implementation under Scilab

3.2.1 Designing the block diagram

First of all, we have to design the system using SCICOS. Figure 1 represents the Scicos block diagram of the given example. 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 block diagram.

- I/O can be chosen from a specific RTAI Library (SCICOS Menu- Edit - Palettes - RTAI-Lib)
- I/O were configured by hand (SCICOS Menu - Edit - Add new block - *Get block GUI from function name* - Name: ....). See next sections how-to make a new user I/O block.

These methods can be mixed together.

3.2.2 Implementing I/O using the RTAI palette

In order to generate the code this block diagram must be transformed into a ”Super Block” which can be used to generate the code.

- The menu ”Diagram” ”Region to Super Block” allows to select a part of the block diagram and put it into a ”Super Block”. Figure 2 represents ”clock” and ”Super Block”.
- We can access to the ”Super Block” block diagram simply by clicking on it (figure 3). 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” - ”RTAI Code Gen” to generate and compile the realtime code.
- Inside ”RTAI” there is a second optional ”Set Target” submenu which activate a dialog box that allow to chose ”Target”, ”Ode function” and ”Step beteween sampling”. Leave the default values.
- A dialog Box asks about some compilation parameters, in particular it proposes the sampling time read from the ”clock” event block.
- After ”OK” SCICOS performs the code generation and the compilation of the generated modules.

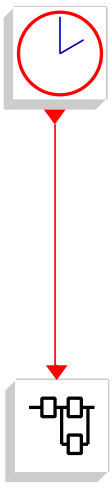

Figure 2: Scicos superblock and clock

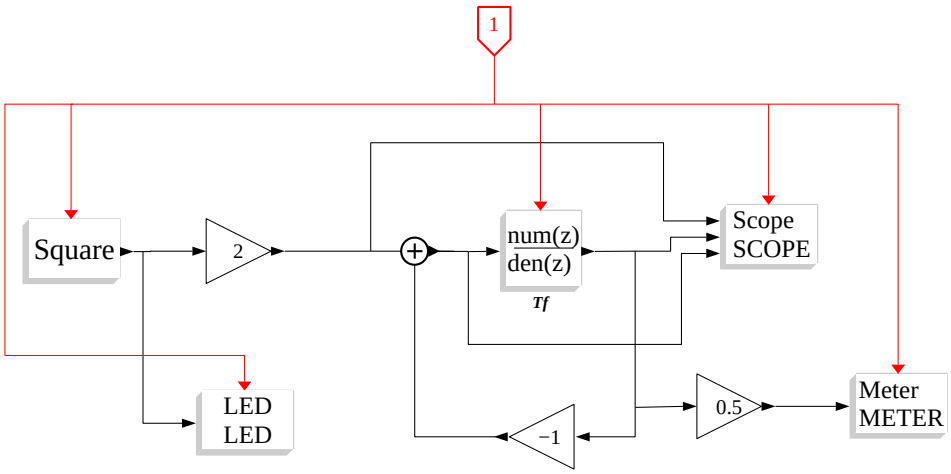

Figure 3: Scicos superblock

4 RTAI-Lib Blocks

Figure 4 represents the Scicos RTAI blocks library.

- Sine. Realtime internal sine generator. Every parameter is programmable and run-time controllable with RTAI-Lab.
- Square. As above but square waveform.
- Step. As above, but step function.
- extdata. Data from external file. The passed file must contain a single column with the values at each sampling time in ASCII format.
- SENSOR. Generic sensor input. The dialog box allows to implement the C-Code.
- Scope. Multichannel input oscilloscope.
- Meter. Single channel meter.
- LED. Multichannel LED lamps: switch ON if the input is positive.
- FIFO-O.
- ACTUATOR. Generic actuator output. The dialog box allows to implement the C-Code.
- Mxb Send Ovw.
- Mbx Rcv no blk.
- Mbx Rcv blk.
- Mbx Send if.
- Comedi A/D. Comedi supported Analog Input.
- Comedi D/A. Comedi supported Analog Output.
- Comedi DI. Comedi supported Digital Input.
- Comedi DO. Comedi supported Digital Output.
- SEM wait. This block can be used to synchronize RT tasks using semaphores.
- SEM signal. This block can be used to synchronize RT tasks using semaphores.
- C RTAI. This is a generic block where the user can implement his own C-Code.

The most I/O blocks call functions implemented in a library (libsciblk.a).
Figure 4 shows the palette with the present available RTAI blocks under Scilab/Scicos.

5 Implementation of new User I/O Blocks

The best way to show how to build a new user IO block from zero is to use a very simple example: how to write an output bit and read an input bit of the standard parallel port. To try this example you need a very simple hardware: a DB-25 male connector, a LED and a resistor (figure 5). Check you BIOS setting about the parallel port mode and address. The safest choice is SPP (Standard Parallel Port) and address 0x378. Hardware details are in appendix F.

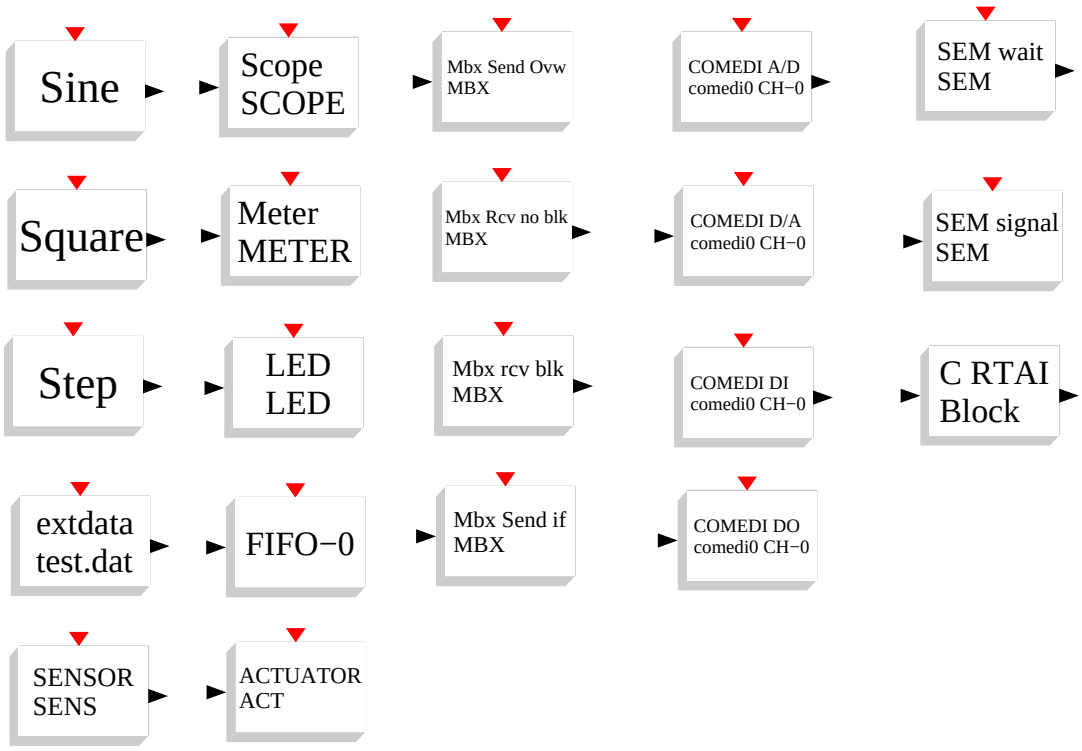

Figure 4: Scicos palette for RTAI

```
Valid addresses are:
         BASE = 0x378  // the standard base address
         BASE = 0x278
         BASE = 0x3BC

For the output line we use (address = BASE)
Position    Name          Pin
B0          Data 0        2

And for the input (address = BASE + 1)
Position    Name          Pin
B3          Error         15
```

5.1 Implementing the code for a user actuator device

- Go to /usr/src/rtai-3.2/rtai-lab/scilab/devices (”cd /usr/src/rtai-3.2/rtai-lab/scilab/devices”)
- Choose a name for the output device (”DirOutBit” Direct Output Bit)
- Use the utilities ”gen_dev <model>” (”gen_dev DirOutBit”). These utilities - using ”template.c” and ”devtmpl.h” - make two very important things: (1) creation of a brand new ”DirOutBit.c” file (”<model>”);this file contains all the necessary functions to implement the driver as ”input” and ”output” driver; (2) updating the ”device.h” files automatically with generic functions prototypes;
- The file ”devstruct.h” contains the description of the structure used to store the block specific data.

```
typedef struct devStr{
  int nch;
```

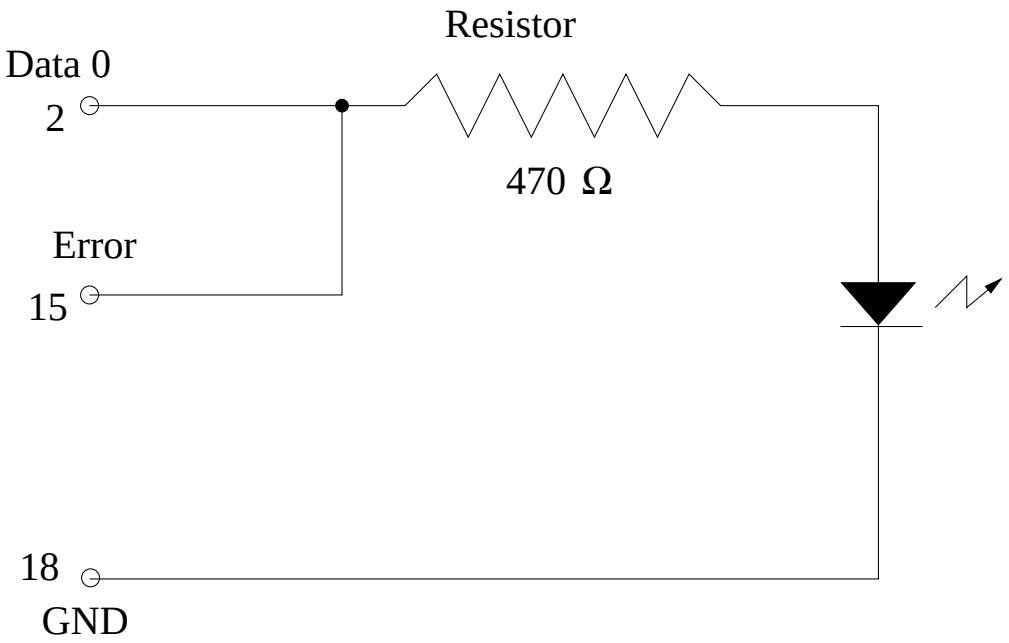

Figure 5: Schematic diagram

```
  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”. The different fields can be used to store temporary data related to the specific instance of the module;

- open with an editor ”GNUmakefile.am” (”gedit GNUmakefile.am” and manually add ”DirOutBit.c” to libsciblk_a_SOURCES list;
- go back to /usr/src/rtai-3.2/ (”cd /usr/src/rtai-3.2/”);
- run here ”aclocal” (this step is optional; use only if you system need it);
- run automake (”automake rtai-lab/scilab/devices/GNUmakefile”);
- make ;
- make install ;

  Now you have an empty code structure ”/rtai-lab/scilab/device/DirOutBit.c”: you need to fill it with some code.
- open ”DirOutBit.c” (”gedit rtai-lab/scilab/device”)
- look carefully to the empty function inside the code ;
- in the top of file add

```
#include <sys/io.h>    The I/O instructions definition
#define BASE 0x378     The parallel port BASE address
```

- pass over the ”inp_xxx_xxx()” these are relative to input block
- ”int out_DirOutBit_init()”. This function is called once the realtime task is started. Clear all the passing arguments of the function and put a ”void”. This is necessary because it is a very simple function without arguments for the ”init” function, then the automatic code generator makes a ”void” call (without any arguments). The ”gen_dev” utility doesn’t know your intentions: it will generate a general empty structure and fill ”device.h” with generic functions prototypes. It is up to you to adjust all the puzzle’s pieces. We put a ”outb (0x00, BASE)” to clear (zeros) all the eight output bit. Make attention to the non-standard syntax of all the I/O instructions (data, address) Normally the I/O port space is protected, but to semplify your job, the code generator uses a special istruction (”iopl(3)” in ”rtmain.c”) to unlock ALL the I/O address space. As usual, this is ”polically incorrect”: you must take all the safety to avoid system crash;
- ”out_DirOutBit_output()”; all the arguments inside the SCICOS realtime environment are pointers to double precision (64 bit) floating point variables. We chose to activate the output bit if the variable is positive;
- ”out_DirOutBit_end()” make the same job of the _init(): clear the output bit ;
- run ”make”; check if the compilation go through without errors or serious warnings then
- run ”make install”

5.2 Creating the SCICOS block for actuator

It it the time for understanding the real guts behind the fancy (or not so fancy) SCICOS graphics block

- go to ”/usr/src/rtai-3.2/rtai-lab/scilab/macros/RTAI” (cd ”/usr/src/rtai-3.2/rtai-lab/scilab/macros/RTAI”)
- copy, customize and save the following examples as ”rtai_DirOutBit.sci”
- open the local ”Makefile” and add to the MACROS list ”rtai_DirOutBit.sci”
- go to ”/usr/src/rtai-3.2/rtai-lab/scilab/macros” (”cd ..”)
- run ”make”

A scicos file specific for an actuator block can be programmed starting from this example. Number $1 \ldots 10$ show the points of the program to be changed (see subsection 5.5).

```
//***************************************************************************************
//                 Here the name of the function must be set.
function [x,y,typ]=rtai_DirOutBit(job,arg1,arg2)                                  // 1
//***************************************************************************************
//
// Copyright roberto.bucher@supsi.ch
x=[];y=[];typ=[];
select job
case 'plot' then
  graphics=arg1.graphics; exprs=graphics.exprs;
//***************************************************************************************
// The user has the possibility to extract some fields which can be used by the ''plot''
// function, in order to display some values on the scicos block here.
  name=exprs(1)(2);                                                              // 2

//***************************************************************************************
  standard_draw(arg1)
case 'getinputs' then
  [x,y,typ]=standard_inputs(arg1)
case 'getoutputs' then
```

```
  [x,y,typ]=standard_outputs(arg1)
case 'getorigin' then
  [x,y]=standard_origin(arg1)
case 'set' then
  x=arg1
  model=arg1.model;graphics=arg1.graphics;
  label=graphics.exprs;
  while %t do
//*****************************************************************************************
// This is the dialog box needed to get all the parameters and info of the block.
    [ok,ip,name,lab]=..                                                          // 3
        getvalue('Direct Output Bit D0',..
        ['input ports';
         'Identifier'],..
         list('vec',-1,'str',1),label(1))
//*****************************************************************************************

    if ~ok then break,end
    label(1)=lab
//*****************************************************************************************
//  A name of the generated C-function is generated here.
    funam='o_DirOutBit_' + name;                                                  // 4
//*****************************************************************************************
   xx=[];ng=[];z=0;
    nx=0;nz=0;
    o=[];
    i=[];
    for nn = 1 : ip
      i=[i,1];
    end
    i=int(i(:));nin=size(i,1);
    ci=1;nevin=1;
    co=[];nevout=0;
    funtyp=2004;
    depu=%t;
    dept=%f;
    dep_ut=[depu dept];

    [ok,tt]=getCode(funam)
    if ~ok then break,end
    [model,graphics,ok]=check_io(model,graphics,i,o,ci,co)
    if ok then
      model.sim=list(funam,funtyp)
      model.in=i
      model.out=[]
      model.evtin=ci
      model.evtout=[]
      model.state=[]
      model.dstate=0
//*****************************************************************************************
// Some block parameters can be inserted in the ''rpar'' fields. These parameters can be
// modified by the ''xrtailab'' application.
      model.rpar=[]                                                              // 5
//*****************************************************************************************
      model.ipar=[]
      model.firing=[]
      model.dep_ut=dep_ut
      model.nzcross=0
      label(2)=tt
      x.model=model
```

```
      graphics.exprs=label
      x.graphics=graphics
      break
    end
  end
case 'define' then
  in=1
  insz = 1
//****************************************************************************************
// All the parameters of the dialog box (see point 3) should be initialized here.
  name = 'ACT'                                                                    // 6
//****************************************************************************************

  model=scicos_model()
  model.sim=list(' ',2004)
  model.in=insz
  model.out=[]
  model.evtin=1
  model.evtout=[]
  model.state=[]
  model.dstate=[]
  model.rpar=[]
  model.ipar=[]
  model.blocktype='d'
  model.firing=[]
  model.dep_ut=[%t %f]
  model.nzcross=0

//****************************************************************************************
// The block default values and the block look are set here.
//
  label=list([sci2exp(in),name],[])                                               // 7

  gr_i=['xstringb(orig(1),orig(2),[''DirOutBit'';name],sz(1),sz(2),''fill'');']
//****************************************************************************************
  x=standard_define([3 2],model,label,gr_i)

end
endfunction

function [ok,tt]=getCode(funam)
   textmp=[
          '#ifndef MODEL'
          '#include <math.h>';
          '#include <stdlib.h>';
          '#include <scicos/scicos_block.h>';
          '#endif'
          '';
          'void '+funam+'(scicos_block *block,int flag)';
         ];
  textmp($+1)='{'
  textmp($+1)='#ifdef MODEL'
  textmp($+1)='int i;'
  textmp($+1)='int port;'
  textmp($+1)='double u[' + string(nin) + '];'
  textmp($+1)='double t = get_scicos_time();'
  textmp($+1)='  switch(flag) {'
  textmp($+1)='  case 4:'
//****************************************************************************************
// The initialization code of the block must be programmed here.
```

```
  textmp($+1)='    port=out_DirOutBit_init();'                                                    // 8
//*****************************************************************************************
  textmp($+1)='    break;';
  textmp($+1)='  case 2:'
  textmp($+1)='    for (i=0;i<' + string(nin) + ';i++) u[i]=block->inptr[i][0];'
//*****************************************************************************************
// The input respectively output code of the block must be programmed here.
  textmp($+1)='    out_DirOutBit_output(port,u,t);'                                             // 9
//*****************************************************************************************
  textmp($+1)='    break;'
  textmp($+1)='  case 5:'
//*****************************************************************************************
// The termination code of the block must be programmed here.
  textmp($+1)='    out_DirOutBit_end(port);'                                                    // 10
//*****************************************************************************************
  textmp($+1)='    break;'
  textmp($+1)='  }'
  textmp($+1)='#endif'
  textmp($+1)='}'

  tt=textmp
  ok=%t
endfunction
```

5.3 Some codes may need your attention

With the actual version, it is mandatory to check and adjust some functions details inside ”device_name.c” and ”device.h” (both in ”rtai-lab/scilab/devices”) to match the automatic code generated by SCICOS.
Sometimes we have compilation problems because the - automatically generated - function calling is not coherent with function prototyping (” device.h”) and functions definitions (”device_name.c”) . In case of problems go to the SCICOS generated source folder and run ”make” manually. Open the *.c file and note the calling style. Open both ”device_name.c” and ”device.h” and adjust accordly. The ”gen_dev” utilities don’t know anything about your intentions, then it creates a very generic ”device_name.c” and ”device.h” that could not match the functions calling of the SCICOS automatic code generator.

5.4 Creating the Scicos block for sensor

- Go to /usr/src/rtai-3.2/rtai-lab/scilab/devices (”cd /usr/src/rtai-3.2/rtai-lab/scilab/devices”)
- Choose a name for the output device (”DirInpBit” Direct Input Bit)
- Use the utilities ”gen_dev” (”gen_dev DirInpBit”);
- open with an editor ”GNUmakefile.am” (”gedit GNUmakefile.am” and manually add ”DirInpBit.c” to libsciblk_a_SOURCES list;
- go back to /usr/src/rtai-3.2/ (”cd /usr/src/rtai-3.2/”)
- run here ”aclocal” (this step is optional; use only if your system need it)
- run automake (”automake rtai-lab/scilab/devices/GNUmakefile”)
- run ”make”
- run ”make install”

  Now you have an empty code structure ”/rtai-lab/scilab/device/DirInpBit.c”: you need to fill with some code.
- open ”DirInpBit.c” (”gedit rtai-lab/scilab/device/DirInpBit.c”)

- look carefully to the empty function inside the code ;
- in the top of file add

```
#include <sys/io.h>   ;   The I/O instructions definition
#define BASE 0x378    ;   The parallel port BASE address
```

- in the function ”void inp_DirInpBit_input(double * y, double t)” put the code

```
input_bit = inb(BASE+1) ;
input_bit = input_bit & 0x08 ; // Filter the Bit 3
if ( input_bit > 0) *y = 1.0 ;
  else              *y = 0.0 ;
```

  to read the input ERROR from the parallel port.

- run ”make”; check if the compilation go through without errors or serious warnings then
- run ”make install”

```
//*********************************************************************************************
function [x,y,typ]=rtai_DirInpBit(job,arg1,arg2)                                    // 1
//*********************************************************************************************
//
// Copyright roberto.bucher@supsi.ch
x=[];y=[];typ=[];
select job
case 'plot' then
  graphics=arg1.graphics; exprs=graphics.exprs;
//*********************************************************************************************
  name=exprs(1)(2);                                                                  // 2
//*********************************************************************************************
  standard_draw(arg1)
case 'getinputs' then
  [x,y,typ]=standard_inputs(arg1)
case 'getoutputs' then
  [x,y,typ]=standard_outputs(arg1)
case 'getorigin' then
  [x,y]=standard_origin(arg1)
case 'set' then
  x=arg1
  model=arg1.model;graphics=arg1.graphics;
  label=graphics.exprs;
  while %t do
//*********************************************************************************************
    [ok,op,name,lab]=..                                                              // 3
        getvalue('Set ',..
        ['output ports';
         'Identifier'],..
         list('vec',-1,'str',1),label(1))
//*********************************************************************************************

    if ~ok then break,end
    label(1)=lab
//*********************************************************************************************
    funam='i_DirInpBit_' + name;                                                      // 4
//*********************************************************************************************
    xx=[];ng=[];z=0;
    nx=0;nz=0;
    o=[];
```

```
    i=[];
    for nn = 1 : op
      o=[o,1];
    end
    o=int(o(:));nout=size(o,1);
    ci=1;nevin=1;
    co=[];nevout=0;
    funtyp=2004;
    depu=%t;
    dept=%f;
    dep_ut=[depu dept];

    [ok,tt]=getCode(funam)
    if ~ok then break,end
    [model,graphics,ok]=check_io(model,graphics,i,o,ci,co)
    if ok then
      model.sim=list(funam,funtyp)
      model.in=[]
      model.out=o
      model.evtin=ci
      model.evtout=[]
      model.state=[]
      model.dstate=0
//*****************************************************************************************
      model.rpar=[]                                                              // 5
//*****************************************************************************************
      model.ipar=[]
      model.firing=[]
      model.dep_ut=dep_ut
      model.nzcross=0
      label(2)=tt
      x.model=model
      graphics.exprs=label
      x.graphics=graphics
      break
    end
  end
case 'define' then
  out=1
  outsz = 1
//*****************************************************************************************
  name = 'SENS'                                                                  // 6
//*****************************************************************************************

  model=scicos_model()
  model.sim=list(' ',2004)
  model.in=[]
  model.out=outsz
  model.evtin=1
  model.evtout=[]
  model.state=[]
  model.dstate=[]
  model.rpar=[]
  model.ipar=[]
  model.blocktype='c'
  model.firing=[]
  model.dep_ut=[%t %f]
  model.nzcross=0

//*****************************************************************************************
```

```
  label=list([sci2exp(out),name],[])                                                    // 7

  gr_i=['xstringb(orig(1),orig(2),[''DirInpBit'';name],sz(1),sz(2),''fill'');']
//****************************************************************************************
  x=standard_define([3 2],model,label,gr_i)

end
endfunction

function [ok,tt]=getCode(funam)
   textmp=[
          '#ifndef MODEL'
          '#include <math.h>';
          '#include <stdlib.h>';
          '#include <scicos/scicos_block.h>';
          '#endif'
          '';
          'void '+funam+'(scicos_block *block,int flag)';
         ];
  textmp($+1)='{'
  textmp($+1)='#ifdef MODEL'
  textmp($+1)='int i;'
  textmp($+1)='double y[' + string(nout) + '];'
  textmp($+1)='double t = get_scicos_time();'
  textmp($+1)='  switch(flag) {'
  textmp($+1)='  case 4:'
//****************************************************************************************
  textmp($+1)='    /* Initialisation */'                                                  // 8
//****************************************************************************************
  textmp($+1)='    break;';
//----------------------------------------------------------------------------------------
  textmp($+1)='    case 1:'
//****************************************************************************************
  textmp($+1)='   inp_DirInpBit_input(y, t);'                                           // 9
//****************************************************************************************
  textmp($+1)='    for (i=0;i<' + string(nout) + ';i++) block->outptr[i][0] = y[i];'
  textmp($+1)='    break;'
//----------------------------------------------------------------------------------------
  textmp($+1)='  case 5:'
//****************************************************************************************
  textmp($+1)='    /* end */'                                                            // 10
//****************************************************************************************
  textmp($+1)='    break;'
  textmp($+1)='  }'
  textmp($+1)='#endif'
  textmp($+1)='}'

  tt=textmp
  ok = %t
endfunction
```

5.5 Fitting the code

1. Here the name of the function must be set.
2. The user has the possibility to extract some fields which can be used by the ”plot” function, in order to display some values on the scicos block here.
3. This is the dialog box needed to get all the parameters and info of the block.
4. A name of the generated C-function is generated here.

5. Some block parameters can be inserted in the ”rpar” fields. These parameters can be modified by the ”xrtailab” application.
6. All the parameters of the dialog box (see point 3) should be initialized here.
7. The block default values and the block look are set here.
8. The initialization code of the block must be programmed here.
9. The input respectively output code of the block must be programmed here.
10. The termination code of the block must be programmed here.

The new blocks must be added to the ”Makefile” in the ”macros/RTAI” directory. The user must now simply run ”make” to complete the procedure.

6 Using COMEDI drivers

In order to use COMEDI drivers with the RTAI-Lab environment the following modules should be installed:

- rtai_hal
- rtai_lxrt
- rtai_fifos
- rtai_sem
- rtai_mbx
- rtai_msg
- rtai_netrpc
- rtai_shm
- rtai_comedi
- comedi
- kcomedilib

At this point the user must load the COMEDI specific modules and perform the ”comedi_config” command.

———— APPENDIX ————-

A Configuring a patched kernel for RTAI

From a basic point a view, kernel 2.4.x and 2.6.x have a very similar configuration structure: if you know which item to select, the graphical tool will guide your choice. The compilation steps are strigthforward, as described in the previous sections. The real obstacle for a beginner is to understand the thousands of options in the kernel configuration. We hope that the following guidelines will save them a lot of effort and frustration.

A.1 General Guidelines

- Keep a default safe kernel version
  Keep in the Grub/Lilo menu a copy of the default kernel shipped with the distro.
- What is not present will not fail
  Keep it simple. At first try to select only the options that you really need. Don't care about the angry red message during the boot. You will fix them later, selecting the right option. Your first primary target is to produce a working hard-realtime kernel. Multimedia and other goodies could wait.
- Forget Overclock
  If Linux is a serious operating system, Linux-Adeos-RTAI "hard-as-diamond" realtime operating system is a very serious one, demanding platform. It will push every bit of your hardware to the max. Then, forget about tweaks or overclock. Put the Bios to "safe" (or "default") settings. Use Memtest86 program to test the CPU-Memory subsystem of your PC.
- Hardware incompatibility
  There is a residual probability of hardware incompatibility, because some chipsets implement timer and interrupt controller in a non-canonical way. The fastest way to check your hardware with ADEOS-RTAI is downloading the ISO image of a bootable test CD from www.rtai.org, burning a disk and making a clean boot. If the latency test works your PC is - basically - OK.
- Resident in the kernel or module ?
  Most of the device driver options offer three settings:
  - de-selected (blank)
  - selected as modules (M or a dot (.))
  - selected as permanent code in the kernel.
  If you don't need them leave de-selected: what is not present will not fail. Select as module if your system want to configure dinamically the devices during the boot (as Fedora Core want do with USB) or when you want to control manually the presence of a specific device driver. Otherwise, if you always need the presence of a device (eg. the ATA hard disk) and you want maximum performance put the code in the kernel. Linux is designed as a monolithic kernel. You can criticize it. You can blame it. But it is fast.

A.2 Configuration

These guidelines are for 2.6.x kernel. Now launch the configuration tool ("make xconfig" or "make menuconfig").

- "I see you"
  From the menu "Option" select "Show All Options". You really need this option enabled. The option that you do not see will kill you.
- Learn to read suggestions
  The kernel configuration needs calm, concentration, focus and accurate reading of the suggestions in the bottom right window. If you use the "menuconfig" text program, there is a separe "Help" option for each menu/item.
- Code maturity level options
  Select "Prompt for development ...etc ...".
- General setup
  Leave the default
- Loadable modules support
  Select "Enable module support" and "Automatic module loading" Deselect "Module versioning support". The RTAI modules are not version dependent.

- Processor type and features
  Select here your Sub-architecture (PC-Compatible) and processor Family. Select ”Preemptible kernel”. This option reduces the system latency. It is good for you. Select only the options that you really understand, otherwise leave them unselected. Be sure to deselect ”Use register arguments”.
- Adeos Support
  Leave the default settings
- Power Management
  Modern processors need power management. If you try to disable this feature, your machine is going to be very hot and the BIOS will activate its power management routine anyway. The BIOS routines are awfully slow and have the bad aptitude of disabling ALL interrupts: this is not a good thing in a Linux Realtime system. Please activate the Power Management and ALL the ACPI relevant features for your machine. Leave APM unselected (it is necessary only for VERY old machines). Deselect also CPU Frequency Scaling, if you leave this option active, the timing of the realtime tasks will change dinamically according to the CPU clock.
- Bus options
  Leave the default. Check the support for your hardware. Laptop needs PCCARD (PCMCIA) support.
- Executable File Format
  Leave the default
- Device driver
  Generic driver options: leave the default
- Memory Tecnology Devices (MTD)
  You don’t need them.
- Parallel Port
  Unselect Parallel port support. The standard parallel port is a useful device for realtime debugging and experimenting. You need to leave this resurce free for Comedi or for direct access for kernel and user tasks.
- Plug and Play support
  Leave the default settings.
- Block devices
  Select your devices. Fedora Core III needs also Ram Disk Support and Initial RAM Disk (initrd) to boot.
- I/O Scheduler
  Leave the default
- ATA/ATAPI/MFM/RLL Support
  Select the main item ”ATA/ATAPI/MFM/RLL supportand” all the options relevant to your system. Dont’ be afraid to select too many options: follow the suggestion of the help window.
- SCSI device support
  Make sure the selection of the main item. Leave the others as default. This subdevice needs attention only if you have SCSI devices.
- Old CDROM
  Obsolete devices. Normally all the CD/DVD drives are IDE devices.
- Multi-device support (RAID and LVM)
  You need this option only in special cases. For Fedora Core III default installation you need this option enabled with LVM (Device Mapper) enabled too.
- Fusion MPT support
  Leave this option disabled.

- IEEE 1394 (Firewire)
  Leave disabled.
- IO2 device support
  Leave disabled.
- Network support
  Leave the default. Explore the devices submenu until you find your netword interface. Use ”lspci” to explore your system.
- Amateur Radio, Irda, Bluthoot and ISDN Telephony support
  Leave disabled. You will enable it later.
- Input device
  Make sure of the selection of mouse devices.
- Character devices
  At first try to unselect everythings or leave the defaults. AGP Support and DRI support are the critical items. We will discuss them in the videocard section.
- I2C support
  This kind of supervision devices are very useful but create too many problems for ADEOS in the handling of their interrupt routine. Leave this option disabled.
- Dallas 1 wire bus, Mish devices
  Leave disabled.
- Multimedia devices
  Leave disabled. You will enable them later.
- Graphics support
  Leave it disabled. With this option you can use the advanced features of your video card, but sometimes this creates compatibility problems.
- Console display driver support
  Select VGA text console
- Sound
  Leave disabled. You will enable it later. Prof. Paolo Mategazza, Father and Architect of RTAI, has a PC without soundcard. In 2005 it seems really incredible. In practice, if you work with realtime systems, the soundcard is just a source of unwanted interrupt and DMA activity. On the other hand, Simone Mannori uses a system with Sound Blaster Audigy with 5+1 support and a Sony multichannels amplifier to play ”Matrix” with Xine while controlling several realtime processes.
- USB Support
  Leave disabled or leave the default options. You will refine the details later.
- File Systems
  Select both ext2 and ext3. Most of distributions use them. Suse use RiserFS. Leave untouched the other options.
- CD/ROM-DVD Filesystem
  Select ISO9660
- DOS/FAT/NTFS
  Select what you need.

  Leave the other menu and options as default.

If you recompile the kernel to add some features, you need also to recompile, install and check both RTAI and Comedi.

B Configuring boot manager

We report a dump of a ”/etc/grub.conf” for a Fedora Core II system as generic reference.

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd0,1)
#          kernel /vmlinuz-version ro root=/dev/hda3
#          initrd /initrd-version.img
#          boot=/dev/hda
default=2
timeout=10
splashimage=(hd0,1)/grub/splash.xpm.gz
#
# The original F.C. II Generic-Modular Non-Realtime Kernel
title Fedora Core (2.6.5-1.358)
root (hd0,1)
kernel /vmlinuz-2.6.5-1.358 ro root=LABEL=/ rhgb quiet
initrd /initrd-2.6.5-1.358.img
#
# The latest ADEOS patched Kernel
title Linux (2.6.10-RTAI-3.2)
root (hd0,1)
kernel /vmlinux-2.6.10-Adeos
#
# Old series kernel
title Linux (2.4.27-RTAI)
root (hd0,1)
kernel /vmlinux-2.4.27-RTAI
#
# VERY useful memory tester program
title Memtest86
root (hd0,1)
kernel /memtest86+-1.11
#
# If you work in industry like me, you may need this bunch of software :(
title Windows XP-Home
rootnoverify (hd0,0)
chainloader +1
```

C Installing 3D DRI support inside a 2.4.x kernel

We describe the MTRR / AGP / DRI configuration options.

- Processor type and features
  Select MTRR support
- Character device - dev/agpgart (AGP support)
  Select your chipset. Inserting a chipset which is NOT present it is not - generally - dangerous because the corresponding driver is activated ONLY if it is correctly detected.
- Character device - Direct Rendering Manager
  Open the submenu and go to ”DRM 4.1 Driver” (in case, unselect the old one). Select your video card. Some cards have a better support than others (ATI are the best ones, refer to DRI web pages for detailed information).

D Installing ATI proprietary driver for kernel 2.6.x

The ATI driver is splitted into two sections: (1) a X11 (Xorg or XFree) driver and (2) a 3D kernel module ”accelerator”. If you have an ATI supported videocard is a good idea to use the ”fglrx-config” program to configure your machine to use the proprietary video driver. In this case the machine will be not 3D accelerated but the proprietary driver is definetely better.

- Download the driver form www.ati.com and unpack with
- ”rpm $--$install $--$force $<$ .rpm$>$”
- configure your diplay using ”fglrx-config”

The X11 driver is configured in ”/etc/X11/xorg.conf”

```
Section "Device"
    Identifier                          "ATI Graphics Adapter"
    Driver                              "fglrx"
```

The ATI custom X11 driver is a very good choice anyway, beacause the generic X11 ”radeon” driver shipped with Xorg/XFree is not very welcome for the latest ATI models.

```
Section "Device"
Identifier  "Videocard0"
Driver      "radeon"
VendorName  "Videocard vendor"
BoardName   "ATI Radeon Mobility 9600 M10"
EndSection
```

To fully use the 3D-OpenGl acceleration you really need to compile a brand-new (Adeos patched of course) 2.6.x Linux kernel with the following mandatory options:

- in section ”Processor type and features”: disable ”MTRR” support ;
- in section ”Character devices”: disable both ”dev/agpgart (AGP Support)” and ”Direct Rendering Manager”
- save the new ”.config” and compile the new kernel. These options must be disabled because the ”fglrx” kernel module needs direct access to the hardware;
- reboot the machine with the new kernel;
- make a safe copy of your xorg.conf (you never know : ”cp /etc/X11/xorg.conf /etc/X11/xorg.conf.backup”)
- launch ”fglrxconfig”
- answer all the questions after reading the text. Normally the suggested default is correct.
- change working directory to

  ```
  cd /lib/modules/fglrx/build_mod/
  sh build.sh" --> (build a brand-new 3D acceleration module for you kernel)
  cd ..
  sh make_install.sh" --> (install the new kernel fglrx module)
  ```

- Open a text console [CRTL]+ALT+[F1];
- login as ”root”

  ```
  init 3
  init 5
  ```

to shutdown and reboot the X server.

- now ”lsmod” signal the presence of a new kernel module

```
Module                  Size  Used by
fglrx                 238716  9
ohci_hcd               21512  0
ehci_hcd               30852  0
```

- Verify the new setup using

```
glxinfo
glxgears
```

and

```
fglrxinfo
fgl_glxgears
```

If everything works OK you are ready to play Doom, Quake or whatever you like while playing DVD, surfing the Web and running hard realtime tasks, of course.

E Code Examples

We report the printout of some configuration and code files.

E.1 GNUmakefile.am

Device drivers makefile.

```
moduledir = $(DESTDIR)@RTAI_MODULE_DIR@

CROSS_COMPILE = @CROSS_COMPILE@

lib_LIBRARIES = libsciblk.a

libsciblk_a_SOURCES = \
rtai_scope.c \
rtai_led.c \
rtai_meter.c \
rtai_fifo.c \
extdata.c \
mbx_receive_if.c \
mbx_receive.c \
mbx_ovrwr_send.c \
mbx_send_if.c \
rtmain.h \
rtai_sem.c \
cioquad4.c \
        maxon_can.c \
        libpcan.c \
rtai_epp.c \
libpcan.h \
pcan.h \
rtai_pport.c\
        mio_mio.c\
        nuke.c\
```

```
DirOutBit.c \
        DirInpBit.c


if CONFIG_RTAI_COMEDI_LXRT
libsciblk_a_SOURCES += \
rtai_comedi_data.c \
rtai_comedi_dio.c
endif

libsciblk_a_AR = $(CROSS_COMPILE)ar cru

includedir=$(prefix)/include/scicos

include_HEADERS = \
devices.h \
devstruct.h

INCLUDES = \
@RTAI_USER_CFLAGS@ \
-I$(top_srcdir)/base/include \
-I../../../base/include \
-I$(top_srcdir)/addons/comedi \
-I@COMEDI_DIR@/include

EXTRA_DIST = template.c README.devices devtmpl.h cioquad4.c
```

E.2 DirOutBit.c

The source code of the Direct Output Bit to the parallel port. Observe the code customisation and the elimination of the functions not used.

```
/*
  COPYRIGHT (C) 2003  Roberto Bucher (roberto.bucher@die.supsi.ch)
                      Simone Mannori (smannori@f2n.it)

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
*/

#include <stdio.h>
#include <string.h>
#include "devstruct.h"

#include <sys/io.h>
#define BASE 0x378 // The parallel port BASE address

extern devStr inpDevStr[];
```

```
extern devStr outDevStr[];
extern int pinp_cnt;
extern int pout_cnt;

int inp_DirOutBit_init(int nch,char * sName,char * sParam,double p1,
 double p2, double p3, double p4, double p5)
{
  int port=pinp_cnt++;
  inpDevStr[port].nch=nch;
  strcpy(inpDevStr[port].sName,sName);
  strcpy(inpDevStr[port].sParam,sParam);
  strcpy(inpDevStr[port].IOName,"DirOutBit inp");
  inpDevStr[port].dParam[0]=p1;
  inpDevStr[port].dParam[1]=p2;
  inpDevStr[port].dParam[2]=p3;
  inpDevStr[port].dParam[3]=p4;
  inpDevStr[port].dParam[4]=p5;

  return(port);
}

void inp_DirOutBit_input(int port, double * y, double t)
{
  /*     *y=XXXX; */
}

void inp_DirOutBit_update(void)
{
}

void inp_DirOutBit_end(int port)
{
  printf("%s closed\n",inpDevStr[port].IOName);
}

// ------------------------------------------------------------------------

int out_DirOutBit_init(void)
{

  outb(0x00,BASE) ;

  return 0 ;
}

void out_DirOutBit_output(int port, double * u,double t)
{
  /*     XXXX=*u; */

  if ( *u > 0 ) outb(0x01,BASE) ;
      else      outb(0x00,BASE) ;

}

void out_DirOutBit_end(int port)
{
  outb (0x00,BASE) ;
  printf("%s closed\n",outDevStr[port].IOName);
}
```

E.3 device.h

The source code of the header file that contains all the functions prototyping.

```
int out_rtai_scope_init(int nch,char * sName);
void out_rtai_scope_output(int port, double * u,double t);
void out_rtai_scope_end(int port);

int inp_rtai_comedi_data_init(int nch,char * sName, int Range, int aRef);
void inp_rtai_comedi_data_input(int port, double * y, double t);
void inp_rtai_comedi_data_update(void);
void inp_rtai_comedi_data_end(int port);

int out_rtai_comedi_data_init(int nch,char * sName, int Range, int aRef);
void out_rtai_comedi_data_output(int port, double * u,double t);
void out_rtai_comedi_data_end(int port);

int inp_rtai_comedi_dio_init(int nch,char * sName);
void inp_rtai_comedi_dio_input(int port, double * y, double t);
void inp_rtai_comedi_dio_update(void);
void inp_rtai_comedi_dio_end(int port);

int out_rtai_comedi_dio_init(int nch,char * sName,double threshold);
void out_rtai_comedi_dio_output(int port, double * u,double t);
void out_rtai_comedi_dio_end(int port);

int out_rtai_led_init(int nch,char * sName);
void out_rtai_led_output(int port, double * u,double t);
void out_rtai_led_end(int port);

int out_rtai_meter_init(char * sName);
void out_rtai_meter_output(int port, double * u,double t);
void out_rtai_meter_end(int port);

int inp_extdata_init(int nch,char * sName);
void inp_extdata_input(int port, double * y, double t);
void inp_extdata_update(void);
void inp_extdata_end(int port);

int out_mbx_ovrwr_send_init(int nch,char * sName,char * IP);
void out_mbx_ovrwr_send_output(int port, double * u,double t);
void out_mbx_ovrwr_send_end(int port);

int inp_mbx_receive_if_init(int nch,char * sName,char * IPs);
void inp_mbx_receive_if_input(int port, double * y, double t);
void inp_mbx_receive_if_update(void);
void inp_mbx_receive_if_end(int port);

int inp_mbx_receive_init(int nch,char * sName,char * IP);
void inp_mbx_receive_input(int port, double * y, double t);
void inp_mbx_receive_update(void);
void inp_mbx_receive_end(int port);

int out_mbx_send_if_init(int nch,char * sName,char * IP);
void out_mbx_send_if_output(int port, double * u,double t);
void out_mbx_send_if_end(int port);

int inp_rtai_fifo_init(int nch,char * sName,char * sParam,double p1,
                  double p2, double p3, double p4, double p5);
void inp_rtai_fifo_input(int port, double * y, double t);
```

```
void inp_rtai_fifo_update(void);
void inp_rtai_fifo_end(int port);

int out_rtai_fifo_init(int nch,int fifon);
void out_rtai_fifo_output(int port, double * u,double t);
void out_rtai_fifo_end(int port);

int inp_rtai_sem_init(char * sName,char * IP);
void inp_rtai_sem_input(int port, double * y, double t);
void inp_rtai_sem_update(void);
void inp_rtai_sem_end(int port);

int out_rtai_sem_init(char * sNam,char * IPe);
void out_rtai_sem_output(int port, double * u,double t);
void out_rtai_sem_end(int port);

int inp_cioquad4_init(int modul ,char * Addr,int reso, int prec, int Rot, int Reset);
void inp_cioquad4_input(int port, double * y, double t);
void inp_cioquad4_update(void);
void inp_cioquad4_end(int port);

int inp_pcan_init(char * can_id,int Kp, int Ki, int nTyp);
void inp_pcan_input(int port, double * y, double t);
void inp_pcan_update();
void inp_pcan_end(int port);

int out_pcan_init(char * can_id,int Kp, int Ki, int nTyp);
void out_pcan_output(int port, double * u,double t);
void out_pcan_end(int port);

int inp_rtai_epp_init(int nch);
void inp_rtai_epp_input(int port, double * y, double t);
void inp_rtai_epp_update(void);
void inp_rtai_epp_end(int port);

int out_rtai_epp_init(int nch);
void out_rtai_epp_output(int port, double * u,double t);
void out_rtai_epp_end(int port);
void rtai_epp_outb(double value, int port);

void inp_mio_mio_init(int port,int nch,char * sName,char * sParam,double p1,
                  double p2, double p3, double p4, double p5);
void inp_mio_mio_input(int port, double * y, double t);
void inp_mio_mio_update(void);
void inp_mio_mio_end(int port);

void out_mio_mio_init(int port,int nch,char * sName,char * sParam,double p1,
                  double p2, double p3, double p4, double p5);
void out_mio_mio_output(int port, double * u,double t);
void out_mio_mio_end(int port);


void inp_nuke_init(int port,int nch,char * sName,char * sParam,double p1,
                  double p2, double p3, double p4, double p5);
void inp_nuke_input(int port, double * y, double t);
void inp_nuke_update(void);
void inp_nuke_end(int port);

int out_nuke_init(void);
void out_nuke_output(int port, double * u,double t);
```

```
void out_nuke_end(int port);
//-------------------------------------------------------------------------------
void inp_DirOutBit_init(int port,int nch,char * sName,char * sParam,double p1,
                  double p2, double p3, double p4, double p5);
void inp_DirOutBit_input(int port, double * y, double t);
void inp_DirOutBit_update(void);
void inp_DirOutBit_end(int port);
//-------------------------------------------------------------------------------
int out_DirOutBit_init(void);
void out_DirOutBit_output(int port, double * u,double t);
void out_DirOutBit_end(int port);
//-------------------------------------------------------------------------------
int inp_DirInpBit_init(int port,int nch,char * sName,char * sParam,double p1,
                  double p2, double p3, double p4, double p5);

void inp_DirInpBit_input(double * y, double t);
void inp_DirInpBit_update(void);
void inp_DirInpBit_end(int port);
```

E.4 DirInpBit.c

The source code of the Direct Input Bit from the parallel port. Observe the code customisation and the elimination of the functions not used.

```
/*
  COPYRIGHT (C) 2005  Roberto Bucher (roberto.bucher@die.supsi.ch)
                      Simone Mannori (smannori@f2n.it)

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
*/

#include <sys/io.h>
#define BASE 0x378  //  The parallel port BASE address

extern devStr inpDevStr[];
extern devStr outDevStr[];
extern int pinp_cnt;
extern int pout_cnt;

int inp_DirInpBit_init(int nch,char * sName,char * sParam,double p1,
 double p2, double p3, double p4, double p5)
{
  int port=pinp_cnt++;
  inpDevStr[port].nch=nch;
  strcpy(inpDevStr[port].sName,sName);
  strcpy(inpDevStr[port].sParam,sParam);
  strcpy(inpDevStr[port].IOName,"DirInpBit inp");
  inpDevStr[port].dParam[0]=p1;
```

```
  inpDevStr[port].dParam[1]=p2;
  inpDevStr[port].dParam[2]=p3;
  inpDevStr[port].dParam[3]=p4;
  inpDevStr[port].dParam[4]=p5;

  return(port);
}

void inp_DirInpBit_input(double * y, double t)
{
  int input_bit ;

  /*     *y=XXXX; */

  input_bit = inb(BASE+1) ;

  input_bit = input_bit & 0x08 ; // Filter the Bit 3

  if ( input_bit > 0) *y = 1.0 ;
    else              *y = 0.0 ;
}

void inp_DirInpBit_update(void)
{
}

void inp_DirInpBit_end(int port)
{
  printf("%s closed\n",inpDevStr[port].IOName);
}
```

F Standar Parallel Port

We report some hardare details of SPP as reference.

```
The most basic version of parallel port is the SPP (Standard Parallel Port).

DB-25 Pin allocation.
Pin     Name            Func.    Register      Hardware Inverted
-----|--------------|----------|-----------|----------------------
1       nStrobe        In/Out    Control       Yes
2       Data 0         Out       Data
3       Data 1         Out       Data
4       Data 2         Out       Data
5       Data 3         Out       Data
6       Data 4         Out       Data
7       Data 5         Out       Data
8       Data 6         Out       Data
9       Data 7         Out       Data
10      nAck           In        Status
11      Busy           In        Status        Yes
12      P_Out/_End     In        Status
13      Select         In        Status
14      nAuto          In/Out    Control       Yes
15      Error          In        Status
16      nInit          In/Out    Control
17      nSelect        In/Out    Control       Yes
18-25   Ground         Gnd
-------------------------------------------------------------------
In/Out: TTL level input / Open Collector output
```

```
Out   : TTL level output
In    : TTL level input

Hardware Inverter: the in/out voltage level is inverted respect to
                   normal logic.

I/O BASE valid address: 0x378 - 0x278 - 0x3BC
```

BASE+0 : Data Register

Bit	Pin	Name	Fun.	Register
0	2	Data 0	Out	Data
1	3	Data 1	Out	Data
2	4	Data 2	Out	Data
3	5	Data 3	Out	Data
4	6	Data 4	Out	Data
5	7	Data 5	Out	Data
6	8	Data 6	Out	Data
7	9	Data 7	Out	Data

BASE+1 : Status Register

Bit	Pin	Name	Fun.	Register	Hardware Inverted
0		Reserved			
1		Reserved			
2		Reserved			
3	15	Error	In	Status	
4	13	Select	In	Status	
5	12	P-Out/-End	In	Status	
6	10	nAck	In	Status	
7	1	Busy	In	Status	Yes

BASE+2 : Control

Bit	Pin	Name	Fun	Register	Hardware Inverted
0	1	nStrobe	In/Out	Control	Yes
1	14	nAuto	In/Out	Control	Yes
2	16	nInit	In/Out	Control	
3	17	nSelect	In/Out	Control	Yes
4		En_IRQ(nACK)			
5		En_DATA_READ			
6		unused			
7		unuded			