SCICOS-HIL

Scicos Hardware In the Loop

Simone Mannori, Ramine Nikoukhah, Serge Steer

Project METALAU - Scilab/Scicos Dev. Team

Ver. **11** - **U**nified **L**inux **W**indows – 7 Dec 2006

If you find some blanks pages is because some parts of this document are not yet completed. We plan to release an updated version soon and keep the documentation aligned with the code. Scicos-HDL is a full Open Source project: please help us to improve the documentation quality editing this file and send it back to simone.mannori@inria.fr

An introduction to Scicos

Writing the simulation of an hybrid dynamical system as scripts, using the powerful functions of the Scilab language is possible [1], but it is time consuming and it is very easy to insert bugs during the manual coding. To simplify this task, Scilab includes Scicos [2] a graphical hybrid dynamical system modeler and simulator toolbox. Scicos is used for applications in control systems, communication, signal processing, queuing systems, and to study physical and biological systems. Within Scicos graphical editor it is possible to place, configure and connect blocks, in order to create diagrams to model hybrid dynamical systems, and simulate them.
Most of the Scicos graphical user interface is written in Scilab Language, for complete integration with Scilab, easy customization and maximum flexibility. Scicos is included in Scilab and available for both Windows and Linux. To develop Scicos blocks the binary, pre compiled Scilab/Scicos version is sufficient.

The Scicos blocks internal structure

Each blocks is represented by two functions:

– the interfacing function. Written in Scilab Language, it defines the graphical representation, the input, output and control ports, signals and the user configurable parameters;
– the computational function: the real code used during the simulations. The computational function can be written in Scilab Language, for easy development, or in C, for maximum efficiency and speed. Fortran is

supported but not advisable, because it could be difficult to compile for embedded systems. The C computational function can be pre-compiled or dynamically *compiled-before-simulation*, allowing user's customization without leaving the Scicos environment.

Scicos simulations could be used to interact with real system in many ways:

– Scicos-HIL: (Hardware In the Loop). The Scicos simulator engine can exchange data with real plants using I/O cards and specific Scicos blocks;
– Scicos-Code Generator: internal, general purpose, C generator. External functions/libraries should for the I/O and timing functions;
– Scicos-RTAI: specific code generator for RTAI-Linux [3]. RTAI provide a kernel patch and kernel/user space libraries that add hard real time functionalities. The Comedi project provides standard I/O function.

Some considerations on sampling time

A digital controller require a very accurate and stable time reference, because the position of the poles and zeros of sampling time data systems depend heavily on Ts (sampling time). Any small deviation (called “jitter”, usually less than 10% of Ts) of the sampling instant is equivalent to an “added noise” (disturbances) in the feedback loop.

More substantial deviation (greater than 10% of Ts) can alter the speed/precision performances and, if large enough, destabilize the closed loop system.

For this reason a lot's of effort is put to guarantee the accurate timely execution of the controller's code. Normally, this is done using a hard real time operating system.

Unfortunately, Linux is not an hard real time OS: also with free CPU time there is no guarantee that the sampling time will be respected. The latest 2.6.1x Linux kernel (2.6.18.1, Oct. 2006) , with the low latency features active and with the internal timer set to the maximum resolution (1.0 ms, 1000 Hz) could be considered real-time at 99% for a sampling time of 10ms or greater. This performance is more than enough for educational laboratory applications with electromechanical systems.

The only way to have hard real time performances is configure a patched (RTAI[4], Xenomai[5]) Linux kernel: a quite complex job, because you need also real time drivers for your data acquisition cards. There are situations were hard real time drivers are not yet available (USB devices are typical examples).

Scilab/Scicos real time function should be modified to support the hard real time API system calls used for for the “Realtime” simulation. This exclude the utilization of the “vanilla” standard Scilab/Scicos distribution.

To limit the installation and configuration effort, we decided to follow the “soft real time path” also because we are confident in the further development of the Linux kernel. The next standard Linux kernels will provides further reduction in latency (from milliseconds to microseconds) and better timing mechanism using high resolution timers.

In any case, when the controller diagram is defined, it is possible to use the Scicos RTAI Code Generator to switch to Linux RTAI hard real time environment.

Windows has basically the same soft real time performance of Linux. Hard real time under Windows using commodity, unmodified PC, is possible using proprietary modified kernels. This solutions is not examined here because require a serious (>5000$) investments.

Scicos Hardware In the Loop

With some limitations, it is possible to use Scicos directly to control a real plant.

The main advantage of Scicos-HIL is interactivity: you can run the simulation, tune the regulator and re use the same data/script/diagram directly without leaving the Scilab/Scicos environment.

Soft real time support in Scicos

Within Scicos it is possible to run the simulation in real-time using the menu option [Simulate]-->[Setup], and setting [Realtime scaling] equal to "1". With this parameter set Scicos "wait" for the right time to read the inputs, make the calculations and update the outputs. It is clear that the time required to complete all the actions should be less than the sampling time.

Windows has basically the same real time capability (1ms resolution) and limitations (soft real time only) of the standard Linux. Scicos use Windows specific system call to implement the “Realtime” function. The I/O functions are implemented using Windows specific dll from the board's supplier.

For Linux, real time support is available using standard function calls.

The routines (for both Windows and Linux versions) are enclosed in a single source file “scilab/routines/time/realtime.c”.

We report the code fragment relative at the key Linux function.

We are improving the real time function of both versions.

```
int C2F(realtime)(double *t)
{
  struct timeval now;
  unsigned long long realtime_diff;
  double simulation_diff;
  long long delay;

  if (simulation_doinit) {
    simulation_doinit = 0;
    simulation_start = *t;
  }
  gettimeofday(&now, 0);
  realtime_diff = TIME2ULL(now) - realtime_start;
  simulation_diff = (*t - simulation_start) * simulation_scale;
  delay = (long long)(simulation_diff * 1000000) - realtime_diff;

  if (delay > 0)
      {
    struct timeval d ;
    d = ULL2TIME(delay);
    select(0, 0, 0, 0, &d); //** this is the syscall used to introduce
  }                         //** the wait time
  return 0;
}
```

The computed delay time “delay” is expressed in microsecond, but the Linux kernel internal resolution is 1ms: the processes cannot by precisely delayed (rescheduled) with a real accuracy better than 1ms.

This limit is also intrinsic in all the USB I/O cards. In practice, this timing mechanism is suitable for sampling time greater or equal to 4 ms.

A marginal improvement on the execution can be obtained running Scilab/Scicos as “root” user. As “root”, using the code below, you can change the execution priority of Scilab/Scicos.

```
/* 21 Jun 2006: CAUTION: Gai code inside :) */
int C2F(realtimeinit)(double *t,double *scale)
{
  struct timeval now;

  int ierr      ;
  int policy    ;

  struct sched_param sched_param;
  gettimeofday(&now, 0);
  realtime_start = TIME2ULL(now);
  simulation_doinit = 1;
  simulation_scale = *scale;

  /* */
  // set policy
  policy = SCHED_FIFO ; /* SCHED_FIFO , SCHED_RR , SCHED_OTHER */
  // get max priority
  sched_param.sched_priority = sched_get_priority_max( policy );
  // set scheduler and priority
  ierr = sched_setscheduler(0, policy, &sched_param);
  // check and print
  if (ierr >= 0 )
    printf("I'm [root]: priority = %d \n ", sched_param.sched_priority );
  else
    {
      printf("User Realtime Mode ...  :(  error= %d \n ", ierr);
      printf("... you MUST be [root] to go REALTIME \n");
    }
 return 0;
}

/*  -----------------------------------------*/
int C2F(realtime)(double *t)
{
  struct timeval now;
  unsigned long long realtime_diff;
  double simulation_diff;
  long long delay; //** delay in microseconds

  volatile static unsigned long ovl_count = 0 ;

  if (simulation_doinit) {
    simulation_doinit = 0;
    simulation_start = *t;
  }
  gettimeofday(&now, 0);
  realtime_diff = TIME2ULL(now) - realtime_start;
  simulation_diff = (*t - simulation_start) * simulation_scale;
  delay = (long long)(simulation_diff * 1000000) - realtime_diff;

  if (delay > 0)
    {

       usleep(delay);
    }
  else
    {
      printf ("Realtime  Overload %d  ! \n", ovl_count++ ) ; //** warning
    }

  return 0;
}
```

In any case, the main limitation is the graphics: the X11 server is the responsible. Avoid to use too many scopes on the simulation. Avoid scope windows resize and/or other manipulation, because the user interaction with X11 Scilab windows blocks the simulation. Increase the scope buffers to limit the weight of the X11 calls.

Data acquisition support

Usually the computational function associated with a Scicos block is a C routine. From C is possible to access directly to all the I/O and memory space. This “direct access” technique is not suitable for various reasons: security is the most important one, and compatibility is in the second place.

Using a common function library and a set of custom device drivers (one of each different DAQ board) it is possible to reuse the same unmodified program on different hardware: this is the aim of the Comedi [6] project.

Using the examples found in the Scicos books [1] Cap. 9 ``Scicos blocks'' and Comedi [6] documentation we have developed the interfacing and computational functions.

Data acquisition hardware buses and devices

A short overview of the possible data acquisition interfaces.

ISA 8 and 16 bit cards

The old IBM PC (8 bit) and IBM-AT (16 bit) standard survive in the industrial applications for reliability and low cost reasons.

PC 104

The above standard in a more compact form: used where space and weight are a premium.

PCI

A full 32 bit high speed parallel bus. Most of the recent DAQ boards use this form.

PC-CARD / PCMCIA

Very similar to the ISA 8/16 bit standard, but require special drivers that support the “plug and play” and “hot plug” features, because is possible to insert/remove cards while the PC is switched on and active.

Parallel Port

The basic parallel port can be used, with some limitations, with a digital I/O Comedi driver. Usually the parallel port is used with direct access code.

The parallel port it the optimal solution for home brew devices. We will illustrate some basic examples.

Serial Port

The serial port can be used also with direct access code at the I/O ports of the UART chip.

USB Devices

The USB port is available as two standard: 1.1 and 2.0. The speed are sufficient (11Mbit/s and 480Mbit/s) for standard I/O applications.

Scicos HIL offer support for two kind of USB devices:

- USB-DUX boards: these boards are born with Linux and Comedi as typical application. The Comedi driver support is complete.
- Measurement Computing USB devices. These boards uses the standard HID (Human Interface Standard) protocols. Some nice guys ha written a complete function library to use these boards under Linux.

CAUTION: the USB devices can be used ONLY in soft real time mode. No hard real time (RTAI, Xenomai, etc.) support is available yet because to exchange data with USB devices it is necessary to call Linux kernel functions in the non-hard-real-time domain. To overcome this limitation the full USB communication stack should be ported in the hard-real-time kernel domain (several project are already working on).

Windows Hardware Support

Usually the board arrives with a specific device driver that is installed before plugging in the card. The card's functionalities are used calling some external function shipped as closed source libraries (static lib r dynamic dll).

Linux kernel preparation

A note for Linux RTAI-users

If you have already installed Linux-RTAI and Comedi, these steps can be jumped: Scicos-HIL is FULLY compatible with Scicos-RTAI. The Linux RTAI kernel and the Comedi-RTAI modules are fully compatible with Scicos-HIL. The only difference is that Scicos-HIL works in soft-real time mode ONLY, independently from the presence of Linux RTAI. You can run two applications at the same time if they use different and independent I/O devices.

To obtain the full performances of SCICOS-HIL you need to configure, compile and install manually a “vanilla” kernel. Usually, the kernels shipped with the distributions uses general purpose settings not optimized for real time application. As results, you could not obtain the maximum performances (1ms resolution, 4ms minimum sampling time).

This step can be avoided if the complete source three of you kernel is installed (it is a strict Comedi requirement) and the real time performance of you shipped kernel are sufficient for your application.

The configuration is not critical and more simple than the the hard real time case (RTAI, Xenomai) because you can reuse the standard configuration of your default kernel and is not necessary patch and/or alter the basic settings.

To configure, compile and install a brand new kernel you need some extra package that are automatically installed if you choose the “developer” or “full” configuration option during the installation of your Linux distribution.

We show step-by-step procedure for a Fedora Core III that can be applicable also for other distribution.

From www.kernel.org download a suitable kernel version. Basically any version from 2.6.10 included is valid (the example is for the latest 2.6.18.1)

Open a terminal and switch to root user

```
[simone@buta ~]$ su
Password:
[root@buta simone]#
```

change directory to usr/src

```
[root@buta simone]# cd /usr/src/
```

unpack the tar file

```
[root@buta src]# tar xjvf /home/simone/Kernel/linux-2.6.18.1.tar.bz2
```

change directory

```
[root@buta src]# cd linux-2.6.18.1/
```

copy the default configuration files

```
[root@buta linux-2.6.18.1]# cp /boot/config-2.6.9-1.667 .config
```

run the automatic conversion tool

```
[root@buta linux-2.6.18.1]# make oldconfig
```

answer with [Enter] at all the questions. This step performs the automatic conversion of the configuration file to the new kernel version reusing compatible, equivalent settings.

run

```
[root@buta linux-2.6.18.1]# make xconfig
```

to check the kernel settings (make menuconfig for the text only interface). Be sure to use these settings:

Loadable module support

unset *Module versioning support*

Processor type and features

set *Preemption Model to Preemptible Kernel (Low-Latency Desktop)*.
set *Preempt The Big Kernel Lock*
set *Timer frequency to 1000 Hz*

Left all the others setting untouched. Save and exit.

Run

```
[root@buta linux-2.6.18.1]# make
```

to compile the kernel. Take a break here (from 15 min to hours).

Run

```
[root@buta linux-2.6.18.1]# make modules_install
```

to install the modules

Copy the kernel in the boot directory

```
[root@buta linux-2.6.18.1]# cp arch/i386/boot/bzImage /boot/vmlinux-2.6.18.1
```

Change directory

```
[root@buta linux-2.6.18.1]# cd /boot
```

create the ramdisk init files to boot

```
[root@buta boot]# mkinitrd initrd-2.6.18.1.img 2.6.18.1
```

Open the bootloader configuration files

```
[root@buta boot]# gedti /etc/grub.conf
```

and add the lines

```
........
#
title Linux 2.6.18.1
      root (hd0,0)
       kernel /boot/vmlinux-2.6.18.1 ro root=LABEL=/
      initrd /boot/initrd-2.6.18.1.img
save and exit
.......
```

Reboot the machine and select your kernel

```
[root@buta boot]# reboot
```

Verify that you machine boot correctly and that all the peripherals works. In case of problems you need to adjust the kernel configuration, recompile, re-install and re-boot.

When the kernel is OK you can pass to the Comedi installation.

Comedi installation

Switch to “root” with the “su” command; go to “/usr/src”

```
[root@buta simone]# cd /usr/src/
```

Create the “Comedi” directory

```
[root@buta simone]# mkdir Comedi
```

Download there the latest CVS snapshot of Comedi and Comedilib from “www.comedi.org” using your favorite web browser.

Unpack the two tar files:

```
[root@buta Comedi]# tar xzvf /home/simone/ComediCVS/comedi.tar.gz
```

and

```
[root@buta comedi]# tar xzvf /home/simone/ComediCVS/comedilib.tar.gz
```

Change directory

```
[root@buta comedi]# cd comedi
```

Create the configure file

```
[root@buta comedi]# sh autogen.sh
```

Run the auto configuration tool

```
[root@buta comedi]# ./configure
```

Compile the drivers

```
[root@buta comedi]# make
```

Install the drivers

```
[root@buta comedi]# make install
```

Create the inodes /dev/comedi0 /dev/comedi15

```
[root@buta comedi]# make dev
```

For Comedilib the procedure is quite similar

```
[root@buta comedi]# cd comedilib
[root@buta comedi]# sh autogen.sh
[root@buta comedi]# ./configure
```

On Fedora Core III you need to download “comedilib-0.7.22” stable version and disable Ruby binding with:

```
[root@buta comedilib-0.7.22]# ./configure  --disable-ruby-binding
[root@buta comedi]# make
[root@buta comedi]# make install
```

Use Comedi device drivers

ISA, PC104, PCI and parallel port device drivers basically use the same installation mechanism described here for the PCI-DAS1602/16.
PC-CARD (PCMCIA) and USB devices require some special attention.

PCI-DAS1602/16

The PCI- DAS1602/16 is a general purpose, multi functions data acquisition cards (DAQ card) with 16 differential/single ended analog inputs (16 bits, 200kHz sample rate), two analog output (16 bits, 100kHz update rate), 24 bits programmable digital I/Os and three 16 bit down counters.
To use a Comedi-compliant card you need a scrip with just two lines

modprobe -v <name_of_the_driver>

and

comedi_config <inode> <name_of_the_driver>

Create a little script that install the driver and configure the board

[root@buta Comedi]# gedit pcidas

```
...
# These lines are only for UDEV systems that require the inode re-creation
# after a boot
for i in `seq 0 15`; do \
        rm /dev/comedi$i
        mknod -m 666 /dev/comedi$i c 98 $i \
        ; \
done;
# If your system does not use UDEV you don't need the above lines
# install the card's driver
modprobe -v cb_pcidas
# configure the board and “mount it” as /dev/comedi0
comedi_config /dev/comedi0 cb_pcidas
...
```

Save the script and with

[root@buta Comedi]# chmod 777 pcidas

to let be executable.

It is important to check the correct hardware/software installation: go there

[root@buta simone]# cd /usr/src/Comedi/comedilib-0.7.22/demo/

and run “info”

[root@buta demo]# ./info

“info” produces a lot of interesting informations about the card

```
 [root@buta demo]# ./info
overall info:
  version code: 0x000749
  driver name: cb_pcidas
  board name: pci-das1602/16
  number of subdevices: 7

subdevice 0:
  type: 1 (analog input)
  number of channels: 16
  max data value: 65535
  ranges:
    all chans: [-10,10] [-5,5] [-2.5,2.5] [-1.25,1.25] [0,10] [0,5] [0,2.5]
[0,1.25]
  command:
    start: now|ext
    scan_begin: follow|timer|ext
    convert: now|timer|ext
    scan_end: count
    stop: none|count
  command fast 1chan:
    start: now 0
    scan_begin: follow 0
    convert: timer 5000
    scan_end: count 1
    stop: count 2

subdevice 1:
  type: 2 (analog output)
  number of channels: 2
  max data value: 65535
  ranges:
    all chans: [-5,5] [-10,10] [0,5] [0,10]
  command:
    start: int
    scan_begin: timer|ext
    convert: now
    scan_end: count
    stop: none|count
  command fast 1chan:
    start: int 0
    scan_begin: timer 10000
    convert: now 0
    scan_end: count 1
    stop: count 2

subdevice 2:
  type: 5 (digital I/O)
  number of channels: 24
  max data value: 1
  ranges:
    all chans: [0,5]
  command:
    not supported


subdevice 3:
  type: 8 (memory)
  number of channels: 256
  max data value: 255
  ranges:
    all chans: [0,1]
  command:
    not supported

subdevice 4:
  type: 9 (calibration)
  number of channels: 8
  max data value: 255
  ranges:
    all chans: [0,1]
  command:
    not supported

subdevice 5:
  type: 9 (calibration)
  number of channels: 2
  max data value: 255
  ranges:
    all chans: [0,1]
  command:
    not supported

subdevice 6:
  type: 9 (calibration)
  number of channels: 1
  max data value: 255
  ranges:
    all chans: [0,1]
  command:
    not supported
```

For our applications the most important informations are:

```
subdevice 0:  type: 1 (analog input)
  number of channels: 16
  max data value: 65535
  ranges: all chans:
[-10,10] [-5,5] [-2.5,2.5] [-1.25,1.25] [0,10] [0,5] [0,2.5] [0,1.25]
0           1     2           3           4     5     6           7

subdevice 1: type: 2 (analog output)
  number of channels: 2
  max data value: 65535
  ranges: all chans:
[-5,5] [-10,10] [0,5] [0,10]
0           1     2     3

subdevice 2:
  type: 5 (digital I/O)
  number of channels: 24
  max data value: 1
  ranges: all chans: [0,5]
```

Scicos-HIL

Unpack “ScicosHil.tar.gz”

Our basic package is composed by four Scilab interfacing functions and the four associated C computational functions

comedi_an_in.sci	*comedi_analog_input.c*
comedi_an_out.sci	*comedi_analog_output.c*
comedi_dig_in.sci	*comedi_digital_input.c*
comedi_dig_out.sci	*comedi_digital_output.c*

the Scilab build script

build_shared_lib.sce

and the startup Scilab script

go.sce

Go inside the Comedi directory

Copy the Comedi shared library “libcomedi.so” from “/usr/local/lib/” and the “scicos_block.h” from “SCI/routines/scicos/.

The build script

A single Scilab script “*build_shared_lib.sce*” built all the project.

The key function is “ilib_for_link” utility for shared library management with automatic link.

libn = ilib_for_link(comp_fun, prog_lst, libs, flag, makename, loadername, libname)

Parameters

comp_fun: a string matrix giving the entry names which are to be linked. The most convenient solution is create a single file for each computational functions, that contain a single main function (and some sub-functions if required). The main function name and the filename (use the *.c extension) should be the same.

c_prog_lst: string matrix giving objects files needed for shared library creation. This script catch ALL the file with *.c extension in the local directory.

libs: string matrix giving extra libraries needed for shred library creation. In this case the Comedilib "libcomedi".

flag: a string flag ("c" or "f") for C or Fortran entry points.

makename: character string. The pathname of the Makefile file without extension (default value “Makelib”). This utilities create a standalone makefile “Makelib” that can be used to create the project at the command line instead that inside Scilab. Could be useful for debugging purposes.

loadername: character string. The pathname of the loader file (default value is “loader.sce”). The automatic Scilab script that load the library.

libname: optional character string. The name of the generated shared library “scicoshilcomedi”.

The complete “build_shared_lib.sce” is

```
//**
comp_fun =
['comedi_analog_input','comedi_analog_output','comedi_digital_output',
'comedi_digital_input'] ;
c_prog_lst = listfiles('*.c');
prog_lst = strsubst(c_prog_lst, '.c', '.o');
libs = "libcomedi" ; //** external Comedi library
flag = "c"         ; //**
makename = 'Makelib';
loadername = 'loader.sce';
libname ='scicoshilcomedi';
libn = ilib_for_link(comp_fun, prog_lst, libs, flag, makename, loadername,
libname);
```

The execution of this script produces this output

```
--> comp_fun  =


         column 1 to 3

!comedi_analog_input  comedi_analog_output  comedi_digital_output  !

         column 4

!comedi_digital_input  !
 c_prog_lst  =

!comedi_analog_input.c    !
!                         !
!comedi_digital_input.c   !
!                         !
!comedi_digital_output.c  !
!                         !
!comedi_analog_output.c   !
 loadername  =

 loader.sce
 libname  =

 scicoshilcomedi
   generate a loader file
   generate a Makefile: Makelib
   running the makefile
   compilation of comedi_analog_input
   compilation of comedi_digital_input
   compilation of comedi_digital_output
   compilation of comedi_analog_output
   building shared library (be patient)
 libn  =

 libscicoshilcomedi.so


-->
```

The “go.sce” script

```
//**
exec("loader.sce");

exec('comedi_an_in.sci');
exec('comedi_an_out.sci');
exec('comedi_dig_out.sci');
exec('comedi_dig_in.sci');
```

load the shared library “libscicoshilcomedi.so” and the Scilab interfacing functions inside the Scilab workspace.

Loading Scicos-HIL blocks inside a Scicos diagram
Now you can run Scicos with

```
-->scicos();
```

The ScicosHIL blocks can be added at the diagram using the “Edit”->”Add new block” using the name of the interfacing function (“comedi_an_in”, “ comedi_an_out”, “comedi_dig_in”, “comedi_dig_out.sci”).

A more useful way is to load, one time for all, all the Scicos-HIL blocks and use “Diagram”->”Save as Palette” with “ScicosHIL” as name: a new palette will be created and the blocks will be copied from it in the usual way.

Please, don't forget to put “1” inside the “Simulate”->”Setup”->”Realtime scaling”.

PC-CARD DAS16/16-AO

The PC-CARD (PCMCIA) needs a special configuration procedure because the card driver's should be loaded automatically - by the PC-CARD daemon – at the at the plug-in: the PC-CARDs drivers should be able to handle the “hot plug” of the devices.

USB-DUX

USB-DUX is fully supported by a Comedi driver. The installation script is just a bit different because the USB-DUX uses a on board micro controller that require some firmware after the hot plug.

Create a little script that install the driver and configure the board

```
[root@buta Comedi]# gedit dux
...
# These lines are only for UDEV systems that require the inode re-creation
# after a boot
for i in `seq 0 15`; do \
        rm /dev/comedi$i
        mknod -m 666 /dev/comedi$i c 98 $i \
        ; \
done;
# If your system does not use UDEV you don't need the above lines
#
# install the card's driver
modprobe -v usbdux
#
# load the firmware, configure the board and “mount it” as /dev/comedi0
comedi_config -i /usr/local/share/usb/usbdux_firmware.hex /dev/comedi0 usbdux
...
```

Save the script and with

```
[root@buta Comedi]# chmod 777 dux
```

to let be executable.

Measurement Computing USB devices under LINUX

All the Measurement Computing USB cards uses the HID (Human Interface Device) standard low level USB driver. This driver is normally included in all the standard kernel shipped with the major Linux distributions that use 2.6.x.

If the board is not recognized, you should verify the kernel configuration (see “Linux kernel preparation”).

From the Linux Kernel Configuration (“make xconfig”) help window:

USB Human Interface Device (full HID) support (USB_HID)

Say Y here if you want full HID support to connect keyboards, mice, joysticks, graphic tablets, or any other HID based devices to your computer via USB. You also need to select HID Input layer support (below) if you want to use keyboards, mice, joysticks and the like ... as well as Uninterruptible Power Supply (UPS) and monitor control devices.

You can't use this driver and the HIDBP (Boot Protocol) keyboard and mouse drivers at the same time. More information is available: <file:Documentation/input/input.txt>.

If unsure, say Y.

To compile this driver as a module, choose M here: the module will be called ***usbhid***

The “usbhid” is a low level, universal driver: to use a real device you need a device specific library. An independent developer (Warren Jasper) has released a complete software suite with some documentation and working examples. We have chosen the USB-1208FS because it have a good price/performance ratio; for different devices you need to adjust some files.

The libraries files are “pmd.c” and “usb-1208FS.c”; “pmd.c” is common to all the cards, “usb-1208FS.c” is specific to the model used. You don't need to edit these files.

As usual, I suggest to work as “root” user.

“Makefile” is the script used to automatically create the library and the test program

#make clean

clear the project

#make

compile all the library files

#make install

install the library (static and shared libs in "*/usr/local/lib*".

Please verify if "/usr/local/lib" is present in the list of path for shared library with

```
#cat /etc/ld.so.conf
```

```
include ld.so.conf.d/*.conf
/usr/X11R6/lib
/usr/lib/qt3/lib
/usr/local/lib                    <------ Yes
/usr/lib
/usr/local/include/efltk
/usr/local/include
```

If is not present, please add the path using an editor

```
#gedit /etc/ld.so.conf
```

and run

```
#ldconfig
```

Now you can connect the board. Check if the green LED in the center light up.

Then check with "dmesg" if the board is configured:

```
#dmesg
..........
hiddev0: USB HID v1.10 Device [MCC USB-1208FS] on usb-0000:00:1d.3-2
hiddev1: USB HID v1.10 Device [MCC USB-1208FS] on usb-0000:00:1d.3-2
hiddev2: USB HID v1.10 Device [MCC USB-1208FS] on usb-0000:00:1d.3-2
hiddev3: USB HID v1.10 Device [MCC USB-1208FS] on usb-0000:00:1d.3-2
usbcore: registered new driver xpad
drivers/usb/input/xpad-core.c: driver for Xbox controllers with mouse
emulation v0.1.4
```

The first four lines means that all the internal sub-devices of the board (analog inputs, analog outputs, digital inputs, digital outputs) are correctly recognized and configured.

The last two lines are specific to my system: some kernels (e.g. Mandrake/Mandriva) recognize the "USB device id" as "Xbox controller".

USB-1208FS has several configuration options for the input/out ports:

– differential analog inputs. The analog inputs can work as differential or singled ended (mixed mode is NOT possible). We choose the four channels differential mode because provide the maximum flexibility;
– analog outputs. The two 0-4.096 Volt analog outputs are not programmable.

We choose the standard 0.000 V as reset output voltage;

- digital I/O. The two (Port A, Port B) digital I/O are bit addressable but each port MUST BE configured as ALL outputs OR inputs: no mixed input/output bit on the same port are allowed (see “usb-1208FS.c” and “usb-1208FS.h” for the details). Please group all the inputs on a port and all the outputs on the other port. Two output ports / two input ports configurations are allowed;

Measurement Computing USB devices under Windows

All the Measurement Computing USB cards uses the HID (Human Interface Device) standard low level USB driver. To avoid compatibility problems we strongly suggest to use latest Windows XP (Home/Media Center/Professional versions) where the Human Device Interface driver is present, automatically loaded and configure at the plug in of the cards.

To use the code, you can simply expand the files in a folder and run the demos. For the developer and the curious people we report the instructions to install the free (in the sense of “free beer”) Microsoft tools.

Install Visual C++ Express Edition

Microsoft has released a zero cost version of this C++ compiler.

From this page

http://msdn.microsoft.com/vstudio/express/visualc/default.aspx

go in the download page

http://msdn.microsoft.com/vstudio/express/visualcsharp/download/

Do the installation with the default parameters. We suggest to register the product immediately.

Install “Microsoft ® Windows Server® 2003 R2 Platform SDK Web

This package is required because the VC Express Edition does not contains all the INCLUDE and LIB files required to compile and link program from the command line interface. Download and install the package from this link

http://www.microsoft.com/downloads/details.aspx?FamilyId=0BAF2B35-C656-4969-ACE8-E4C0C0716ADB&displaylang=en

Configure Compiler and SDK

Follow this instructions to configure the development chain

http://msdn.microsoft.com/vstudio/express/visualc/usingpsdk/

Install Scilab
Use the latest binary package (Scilab 4.1) for Windows.

Install the basic MCC DAQ software
The following packages are on the disks that arrives with the data acquisition card.
InstaCalc: this utility is used to guide the hardware configuration of the installed card. The results is a “board.cfg” txt file. This file must be present in the system and in the ScicosHIL folder
TracerDAQ: use this program to verify the installation.

Install the MC Universal Library
This installation prepare a directory with the libraries and some examples source code.

Development of Scicos Interfacing and Computational functions
Unfortunately the “ilib for link” Scilab macros that do most of the low level work does not work well with VC++ Express 2005 edition because the 4.x version are build using .NET platform.
We found a solution using the usual “ilib for link” (that generate some errors), editing by hands the “Makelib.mak” and “loader.sce” files: you can use our reference files as examples.
For compile the computational functions and create the “libmccusb.dll” dinamic library:
run Scilab
launch “*unix start*” at Scilab command line
“unix start” open a “cmd.exe” command line “DOS” interface
In this terminal run
nmake /f Makelib.mak
“nmake” is the Linux “make” equivalent *“/f Makelib.mak”* specify the script files.

References

[1] “Modelling and Simulation in Scilab/Scicos”, S.L. Campbell, JP. Chancellier, R. Mikoukhah, Springer.

[2] www.scicos.org

[3] “RTAI-Lab How To” PDF, av. on line at www.rtai.org (About RTAI-Lab).

[4] www.rtai.org

[5] www.xenomai.org

[6] www.comedi.org