# Advantech Automation Corporation

# Advantech Device Driver for Linux
# User's manual

**Version <2.4>**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 10/28/2002 | <1.0> | Initial version | Bai jingang |
| 11/6/2002 | <1.5> | Update from original comedi manual | Edwin |
| 11/14/2002 | <2.0> | Modified with xinyong | Edwin |
| 12/09/2002 | <2.0.1> | Add supported cards table | Changping |
| 2003/5/1 | <2.1> | Add pci1784, modify installation instructions | Edwin |
| 2003/5/20 | <2.2> | United Linux testing | Edwin du |
| 2004/1/7 | <2.3> | Add pcm3730, add insn convention and special hint of pcm3730 | Eddy Yang |
| 7/15/2004 | <2.4> | Add supported cards table | Zhang dongdong |

# Table of Contents

# User's Manual

## 1. Introduction

Drivers for Advantech data acquisition boards and motion control cards for Linux are developed on COMEDI (Linux Control & Measurement Device Interface).

The COMEDI project develops open-source drivers, tools, and libraries for data acquisition.

COMEDI is a collection of drivers 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.

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

This document describes usage of Advantech Linux drivers with COMEDI in general. Information on specific production please refer to the heading remarks in the driver's source code.

### 1.1 Features

Integrated real-time support for most hardware

High-level library (comedilib)

Application-level device independence

Works with Linux kernel from 2.2.x to 2.4.x.

Driver released together with comedi-0.7.60 and comedilib-0.7.19.

### 1.2 Definitions, Acronyms and Abbreviations

| Definition | Explain |
|---|---|
| COMEDI | Linux Control & Measurement Device Interface |
| AI | Analog input |
| AO | Analog output |
| DI | Digital input |
| DO | Digital output |
| insn | Instruction mode I/O |
| cmd | Command mode I/O |

### 1.3 Overview

Comedi is a collection of Advantech drivers for data acquisition hardware. These drivers work with Linux, and also with Linux combined with the real-time extensions RTAI and RTLinux. The Comedi core, which ties all the drivers together, allows applications to be written that are completely hardware independent.

The distribution of Advantech Linux driver has the following structures:

- Comedi-0.7.60.tar.gz
  This distribution contains just the Comedi kernel modules.
  More information please refer to comedi-0.7.60/README.

Following is the structure of comedi-0.7.60:

```
└──comedi-0.7.60              main COMEDI dir
   ├──Documentation           documentations about install, configure and develop.
   │  └──comedi
   │     └──notes
   ├──comedi                  working dir.
   │  ├──drivers              drivers, including source code and .o files.
   │  └──kcomedilib           kernel space library
   ├──debian
   ├──include                 .h files that COMEDI driver needs.
   │  ├──asm
   │  └──linux
   ├──patches
   ├──rpm
   └──scripts
      ├──linux_flags
      └──lxdialog
```

- Comedilib-0.7.19.tgz
  This distribution contains only the user-space library.
  More information please refer to comedilib-0.7.19/README.
- Test.tar.gz
  This distribution contains testing cases on the driver. Test cases are based on comedilib-0.7.19.

## 1.4 References

- COMEDI Official web site:

http://stm.lbl.gov/comedi or http://www.comedi.org/

- The IO-Port Programming Mini HOWTO:

http://www.linuxdoc.org/HOWTO/mini/IO-Port-Programming.html

- Linux Kernel Module Programming Guide

http://www.linuxdoc.org/LDP/lkmpg/mpg.html

- Kernel Hacker's Guide

http://www.linuxdoc.org/LDP/khg/HyperNews/get/khg.html

- Linux Device Drivers, 2nd Edition

http://www.xml.com/ldd/chapter/book/index.html

- RTAI web site

http://www.rtai.org/ or http://www.aero.polimi.it/~rtai/

# 2. Comedi installation instructions

## 2.1 Comedi-0.7.60 installation

### *2.1.1 Patch for comedi*

When using comedi in RedHat9 or compiling with gcc 3.x.x , you should patch comedi package. Patching process as follow:

1) decompressing file patch.tgz
   **# tar xvfz patch.tgz;**
   The directory construction is:

```
+patch
 |
 +--comedi-0.7.60-gcc3.patch   :patch file for gcc3 problem
 |
 +--comedi-0.7.60-redhat9.patch          :patch file for redhat9 problem
 |
 +--README                                :readme file
```

2) patch for gcc3.x.x
   If your OS (such as Redhat 8.0, 9.0 and Fedora) uses gcc3xx compiler, this patch file is needed. Patch comedi package with the following commands.

   **# ls**
   **comedi-0.7.60/ patch/**

   **# cd comedi-0.7.60/**
   **# patch -p1 < ../patch/comedi-0.7.60-gcc3.patch**

3) patch for redhat 9
   If your OS is Redhat 9.0 with its original kernel, this patch file is needed. Patch comedi package with the following commands.

   **# ls**
   **comedi-0.7.60/ patch/**

   **# cd comedi-0.7.60/**
   **# patch -p1 < ../patch/comedi-0.7.60-redhat9.patch**

More detail please README file

### *2.1.2 Linux source:*

In order to compile the Comedi modules, you will need to have a correctly configured Linux kernel source tree. The best way to get one is to download a tarball from kernel.org and compile your own kernel. Comedi should work with most 2.2 and 2.4 Linux kernels. Support for 2.0.3x is not actively maintained, but it should work and bugs will be fixed as they are reported.

The following steps will (almost) set up your kernel sources correctly. You will also need write permission to the kernel source directory the first time you run comedi's 'make config', so you might want to unpack the kernel source into a directory you own.

There are two ways to prepare Linux kernel source. One is a standard way that we strongly recommend to, which is to download a standard Linux kernel and configure it. The other way informs you the way to configure kernels in some popular Linux release, such as RedHat, Debian, Mandrake, etc.

#### 2.1.2.1 General setup

We strongly recommend to use a standard Linux kernel distributed by http://www.kernel.org , this section describes the most common steps to setup the kernel. This operation is independent to Linux releases. Read the instructions if you have problem compiling the kernels: http://www.tldp.org/HOWTO/Kernel-HOWTO/.

1) Get a copy of the kernel source that matches the kernel you are running. Download a kernel from http://www.kernel.org or from a mirror site nearby.

   The compressed tar archive in this directory can be downloaded using a web browser or a command-line program such as wget:

   ```
   #wget --no-directories --retr-symlinks
    http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.x.x.tar.gz
   ```

2) Copy the package into a directory that you have writes permission and unpack kernel:

   ```
   #cp linux-2.x.x.tar.gz /usr/src
   #cd /usr/src/
   #tar –jxvf linux-2.x.x.tar.bz2   (This command is for .bz2 kernel)
   #tar –zxvf linux-2.x.x.tar.gz    (This command is for .gz kernel)
   ```

3) Configure your kernel:

   Change into your kernel source directory.

   ```
   #cd linux-2.x.x
   #make menuconfig
   ```

   Then you would be provided with a graphic user interface to configure your kernel. Configuration steps is far beyond our scope, please refer to some other manuals.

   After configuration, you should exit and choos 'save' to save your configuration.

   Alternatively, if you already have a configuration file, you could config your kernel by 'make oldconfig':

   ```
   #cp -p YOUR_CONFIG_FILE.config .config
   #make oldconfig
   ```

4) Make dependencies:

   After configuration, you must make dependencies:

   ```
   #make dep
   ```

5) Produce bootable kernel image and get ready to boot using the new kernel:

```
#make bzImage
#make modules
#make modules_install
```

These three steps would compile the source package into a bootable kernel image due to your own configuration, and also install modules into /lib/modules/2.x.x/.

```
#make install
```

This step would copy the target image into /boot/ and setup to boot your new kernel. If you have successfully finished above operations, you might find some new files in /boot/ directory such as vmlinuz-2.x.x and System.map-2.x.x.

6) Append a new item to your bootloader to boot the kernel:

Finally you might be able to boot your new kernel.

In most cases your Linux might use LILO or GRUB as the boot manager, here are some instructions for them respectively.

LILO:

Edit your lilo configuration file to append a new item:

```
#vi /etc/lilo.conf
```

append such new lines to locate your kernel:

```
image=/boot/vmlinuz-2.x.x
label=linux-2.x.x       (any name you want to name it)
root=/dev/hda1          (Here is your '/' location)
read-only
```

Finally you must run lilo to activate your configurations:

```
#/sbin/lilo -v
```

GRUB:

Edit your grub configuration file to append a new item:

```
#vi /boot/grub/menu.lst(this location is where you installed grub)
```

append such new lines to locate your kernel:

```
title linux-2.x.x       (any name)
kernel(hd0,1)/vmlinz-2.x.x root=/dev/hda1
```

7) Reboot system and select the new kernel:

#### 2.1.2.2 Linux distribution kernel Setup

Since each Linux distribution has made some modification to its own kernel package, it might cause some problem to use these kernel sources. Here we provide some suggestions to use these kernels according to each Linux distribution, however, the compiling success is not guaranteed.

In this section kernel packages (quoted as 'linux-2.x.x' following) are ones that released together with specific Linux distributions.

##### 2.1.2.2.1 Redhat

**Red Hat users note: Kernel sources that are distributed with Red Hat Linux are NOT supported, because they are too heavily modified. However, there is some information in Documentation/comedi/redhat-notes on how to use Red Hat kernels.** The following steps are tested under Redhat 7.x, Redhat 8.0 and Redhat 9.0. NOTICE that Redhat 9.0 releases with gcc-3.2.2, which seems to cause some problem when compiling comedilib. You should change gcc version to 3.2.3 or earlier versions with Redhat 9.0.

1) Pre-configuration operation.

```
#cd /usr/src
#ln –s linux-2.x.x linux   (make a symbol link to source dir)
#cd linux
#make mrproper (clear any configuration, must do)
# cp configs/kernel-2.4.18-i686.config .config
```

Copy the correct configuration file to '.config' according to your hardware structure.

```
#make oldconfig
```

Alternatively you could make manuconfig to produce your own .config file

```
#make menuconfig
```

```
#make dep
```

Make dependency.

2) Special modification

Edit /usr/src/include/linux/version.h. This file is generated by 'make dep'. Modify UST_RELEASE item to linux-2.x.x. Originally, there might be some other options as linux-2.x.xcustom, linux-2.x.xsmp,etc. This would cause comedi driver installation problem.

Make sure your system structure:

```
#uname –r
```

This would prompt you your current kernel, modify UST_RELEASE identical to it.

3) Make dependencies again:

```
#make dep
```

#### 2.1.2.2.2 Debian, Mandrake

The kernel package of Debian or Mandrake releases is a standard one which is quite compatible for comedi installation. Following the general way to configure the kernel:

```
#cd /usr/src
#ln –s linux-2.x.x.x linux
#make mrproper
#make menuconfig          (xconfig, oldconfig, config)
#make dep
```

#### 2.1.2.2.3 SuSE, UnitedLinux

Kernel package of SuSE 8.x is a 4-GB memory supported kernel, thus the comedi package should be configured to fit the kernel. It's a hack from mail list of comedi mail list, but it works.

1) Make dependencies of the kernel source, just as the stand way:

```
#cd /usr/src
#ln –s linux-2.x.x.x linux
#make mrproper
#make menuconfig          (xconfig, oldconfig, config)
#make dep
```

2) Modify comedi package

   Edit file comedi/kvmem.h, replace line "#include <asm/pgtable.h>" with "#include <linux/highmem.h>". Otherwise you will probably have the prompt "Unresolved symbol kmap_pagetable" when inserting mode comedi.o.

3) Compile and install comedi pacakge

### *2.1.3 RTAI support:*

If you want to use the real-time capabilities of Comedi with RTAI, you need to compile and install RTAI first. It is necessary to use the rthal patch instead of the "copyto" scripts. Known working versions are RTAI-1.6, RTAI-24.1.4, and current RTAI CVS. Remember to enable Kcomedilib support, since you will be accessing Comedi from other kernel modules.

Comedi mail list would give your more information of RTAI comedi installation.

Advantech drivers are not tested with RTAI, performances of the drivers are not guaranteed.

### *2.1.4 RTLinux support:*

If you want to use the real-time capabilities of Comedi with RTLinux, you need to compile RTLinux (both the kernel and the modules) first. Known working versions are 2.x and 3.0. Remember to enable Kcomedilib support, since you will be accessing Comedi from other kernel modules.

Comedi mail list would give your more information of RTLinux comedi installation.

Advantech drivers are not tested with RTLinux, performances of the drivers are not guaranteed.

### *2.1.5 Configuration:*

First change to a directory that you have write permission.

```
#cd /usr/local/
```

```
#tar –zxvf comedi-0.7.60.tgz
```

This command will depress the comedi into the comedi-0.7.60 directory. Thus the package would be extracted to /usr/local/comedi-0.7.60

After unpack the comedi package, you may configure using '**make config**'. This will ask you the location of the Linux kernel source tree. If it detects that you have a kernel patched for RTAI or RTLinux, it will also ask you for the location of the RTAI or RTLinux source directory. Then the configuration script will ask questions for a couple general Comedi features and then whether or not compile each driver.

```
#make config
```

If you want to reconfigure comedi, use '**make distclean**' and '**make**'.

### *2.1.6 Compiling:*

Compile using '**make**'. If this fails for some reason, send the _entire_ build log to the mailing list. Without the build log, it is impossible to find problems.

```
#make
```

### 2.1.7 *Installation:*

Install using '**make install**' as root. This installs the files:

```
#make install
```

/lib/modules/<<kernel version>>/misc/comedi.o

/lib/modules/<<kernel version>>/misc/kcomedilib.o

/lib/modules/<<kernel version>>/misc/<<driver files>>.o

You need to create device files to access the hardware from a

user process. These can be created using '**make dev**'. The following

special files will be created:

/dev/comedi0

/dev/comedi1

/dev/comedi2

/dev/comedi3

Next you should compile and install comedilib. Refer to section 2.2.

### 2.1.8 *Module Autoloading:*

If you like to autoload your modules, put the following lines into /etc/modules.conf (this does not apply for PCMCIA cards):

alias char-major-98 comedi

alias char-major-98-0 your_driver

post-install your_driver /usr/sbin/comedi_config /dev/comedi0 your_driver <<options>>

Alternatively, for complicated option lists, the scripts in etc are designed to be copied into /etc, so that you could put the following lines into /etc/conf.modules:

alias char-major-98-0 dt282x

post-install dt282x /etc/dt282x.conf

### 2.1.9 *Upgrading:*

From versions prior to 0.6.0, you will need to edit and recompile all programs that use comedi or comedilib, since the names of functions and ioctls have changed.

From versions prior to 0.5.0, you will need to recompile all programs that use comedi or comedilib, since the interface to both of these has changed. No changes should need to be made to the source of the programs. The format for parameters of comedi_config has changed.

From versions prior to 0.4.0, you will need to run 'make dev' again to recreate /dev/comedi*, since the major number has changed.

### 2.1.10 *Un-installation*

Run 'make distclean' to uninstall comedi-0.7.60 from your Linux.

Anyway, you should manually delete some modules located at /lib/modules/2.x.x/comedi/.

#### *2.1.11 Hack*

The newer kernels have the concept of module licenses. Comedi releases do not cover this point. So update amcc5933.c to avoid 'taint the kernel' prompting. Packages provided are configured properly with this, no problem when installing.

If console did not display kernel config messages, type
#echo 8 > /proc/sys/kernel/printk

### 2.2 Comedilib-0.7.19 installation

Compile using '**make**'. If this doesn't work, make sure you have the basic tools installed to compile. If you can successfully compile other things, consult the author, as he has probably made a mistake.

Install using '**make install**' as root. This installs the files:

/usr/lib/libcomedi.so.0.3

/usr/include/comedi.h

/usr/include/comedilib.h

and other things...

if your OS is redhat 9, run '**ldconfig /usr/local/lib/'**

If you run Debian GNU/Linux, comedilib is packaged as the packages libcomedi0 and libcomedi-dev. These are available as part of woody.

## 3. Package configuration

### 3.1 The existent drivers

| Driver name | Supported cards |
| --- | --- |
| **pci1710.o** | **PCI-1710, PCI-1710HG, PCI-1711, PCI-1713, PCI-1720, PCI-1731** |
| **pci1712.o** | **PCI-1712** |
| **pci1716.o** | **PCI-1716** |
| **Pci1721.o** | **PCI-1721** |
| **Pci1730.o** | **PCI-1730, PCI-1733, PCI-1734** |
| **Pci1750.o** | **PCI-1750** |
| **Pci1751.o** | **PCI-1751** |
| **Pci1753.o** | **PCI-1753, PCI-1753E, MIC-3753** |
| **Pci175x.o** | **PCI-1752, PCI-1754, PCI-1756, MIC-3756** |
| **Pci1760.o** | **PCI-1760** |
| **Pci1761.o** | **PCI-1761, MIC-3761** |
| **Pci1762.o** | **PCI-1762** |
| **Pci1780.o** | **PCI-1780** |
| **Pci1784.o** | **PCI-1784** |
| **Pcl711.o** | **PCL-711** |

| | |
| --- | --- |
| **Pcl724.o** | **PCL-724, PCL-722, PCL-731** |
| **Pcl725.o** | **PCL-725** |
| **Pcl726.o** | **PCL-726, PCL-727, PCL-728** |
| **Pcl812.o** | **PCL-812** |
| **Pcl818.o** | **PCL-818** |
| **Pcm3718.o** | **PCM-3718** |
| **Pcm3730.o** | **PCM-3730** |

### 3.2 Adding driver to original comedi

- If a comedi package downloaded from the web site was already installed, a newly written device driver could be added into the comedi environment.
- Put your driver source code into directory comedi-0.7.60/comedi/drivers/.
- To add a driver to comedi, first you should '**make distclean**' to remove original comedi settings.
- Then edit 'comedi-0.7.60/comedi/Config.in' and append a new line, for example:

  ```
  dep_tristate 'PCI-1730' CONFIG_COMEDI_PCI1730 $CONFIG_COMEDI
  ```

  which describes the module dependency for comedi.
- Next you should edit 'comedi-0.7.60/comedi/drivers/Makefile' to add a new make option for the driver. For example:

  ```
  obj-$(CONFIG_COMEDI_PCI1730)            += pci1730.o
  ```

  Remember that the variable here must be consistent to the line in Config.in file.
- Run '**make**' to reconfigure comedi-0.7.60, select the new driver.
- Run '**make**' to compile.
- Then '**make install**' to install the module into Linux.

### 3.3 Running Comedi:

To use comedi, the driver module and the core Comedi modules must be loaded into the kernel. This is done by a command similar to

**/sbin/modprobe <<driver>>**

If your module dependencies are set up correctly, this will load both comedi.o and your driver. If you get unresolved symbols, check the FAQ or the mailing list archives. Also look at the man pages for modprobe and insmod.

To unload the module, run

**/sbin/rmmod <<driver>>**

In order to configure a driver module to use a particular device file (/dev/comediN) and a particular device, you need to use the command /usr/sbin/comedi_config, which is part of the comedilib distribution. Comedi_config is invoked using

**/usr/sbin/comedi_config /dev/comedi0 <device name> <option list>**

The device name may or may not be the same as the module name. In general, if the device type can be auto probed (as with ISA PnP or PCI devices), the device name will be the same as the module name. Otherwise, you will need to check Documentation/comedi/drivers.txt for information about what device name is appropriate for your hardware. The option list（user should partition parameter in option list with comma） is to supply additional information, such as I/O address, IRQ, DMA channels, and other jumper settings. Information about option lists appropriate for a driver is in drivers.txt. The following commands are examples:

/usr/sbin/comedi_config /dev/comedi0 pci1730 0,10

this command combines device /dev/comedi0 to module pci1730, located at pci bus0, slot 10. Remember here we use a device name, so if you want to use pci1733 cards, which are sharing the same driver with pci1730, you should replace 'pci1730' with 'pci1733' in this command.

/usr/sbin/comedi_config /dev/comedi0 pci818 0x300,5,3 –read-buffer 1024

Try a 'man comedi_config' for information on how to use this utility. Scripts have been written for a few of the drivers with very complicated option lists -- these are found in the etc directory.

**For example, if you use only one PCI-1710 card, you should run**

**/sbin/modprobe comedi**

**/sbin/modprobe adv_pci1710**

**/usr/sbin/comedi_config /dev/comedi0 pci1710**

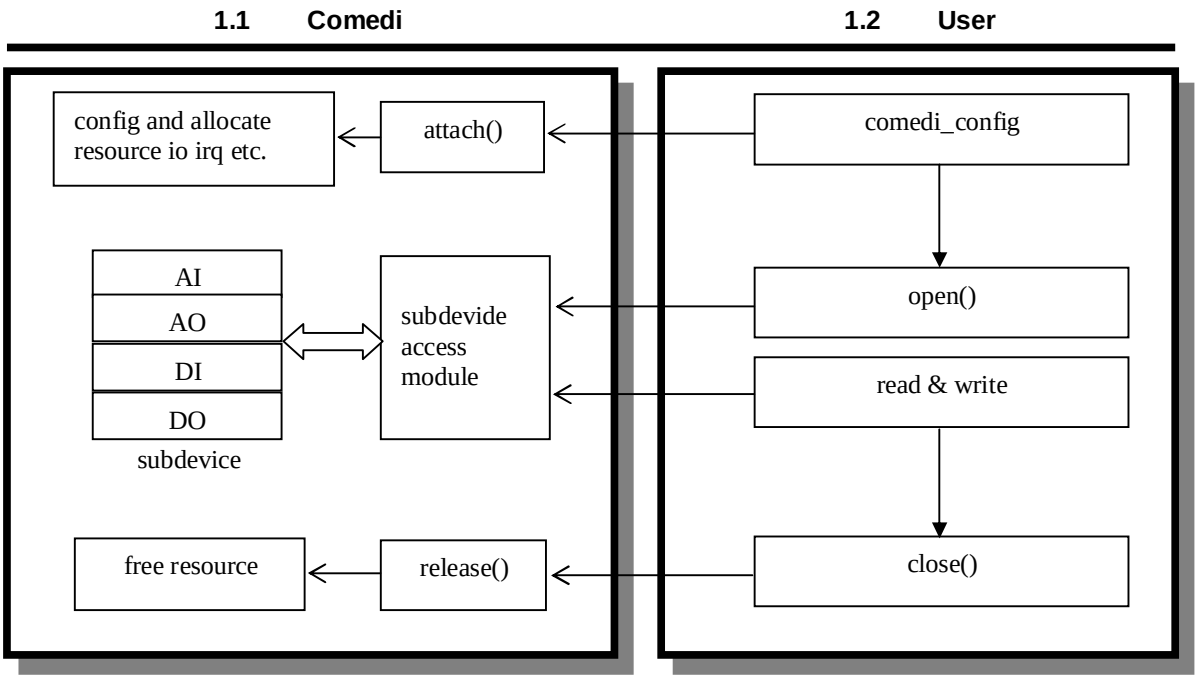
**Attention**: different driver may have different options, so you need to read driver document for more information. At the beginning of each driver's source file, there are the descriptions of how to fill in the above parameters; the users may fill in them accordingly. You can find this information also in 'comedi-0.7.60/Documentation/comedi/drivers.txt', which illustrate it more explicitly.

### 3.4 Multi cards co-existence

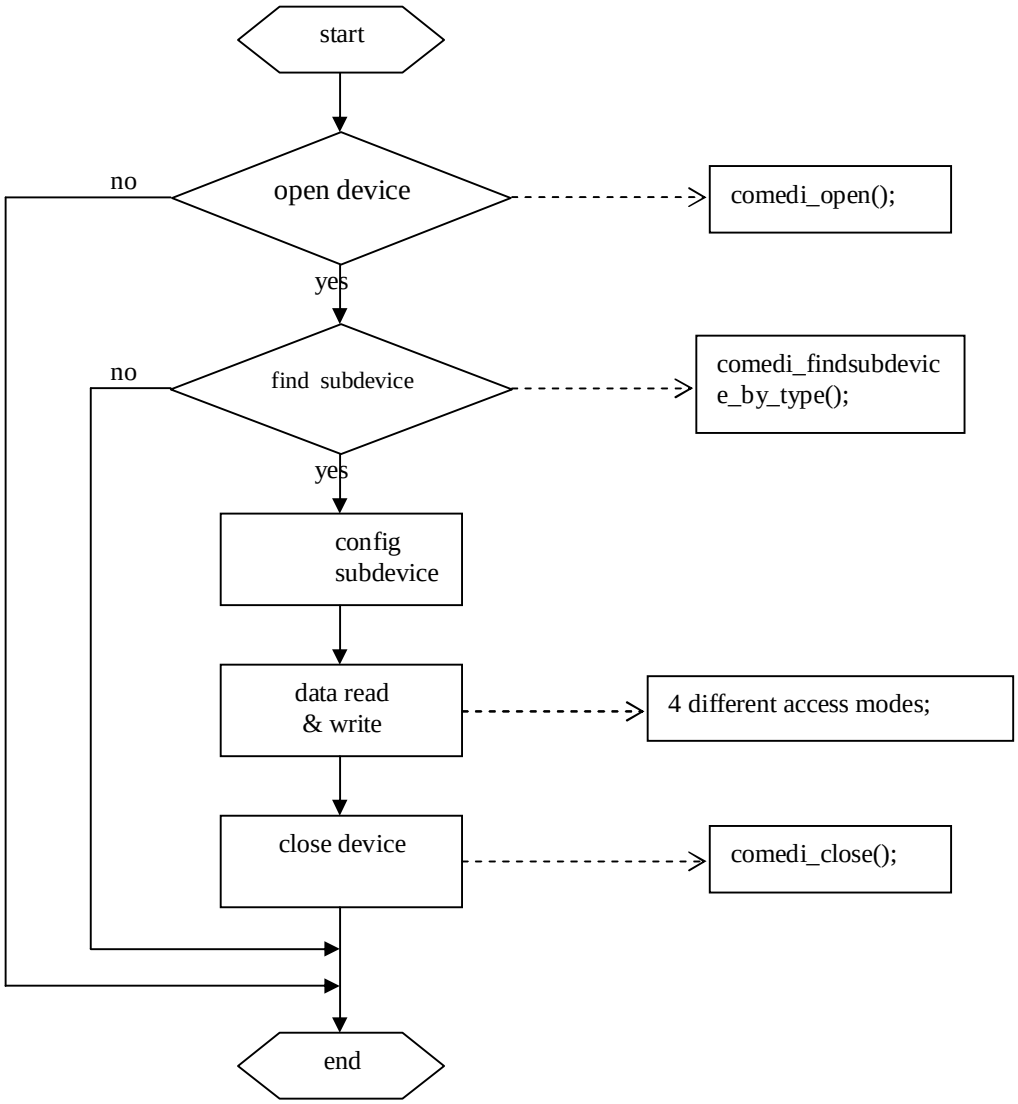
- Device files located at /dev/comediX, by default, there is 4 device files: coemdi0, comedi1, comedi2, comedi3. More device files could be appended manually.
- Comedi_config could bind device file to module pci1730. Specific card could be identified with pci bus and slot, which could be passed through comedi_config command parameters.
- If you have more than 4 card in the same system, you should manually add nodes just as 'make dev' does:
  - **mknod -m 666 /dev/comediX c 98 X**
  - Here 'X' is the number of your nodes, might be 5, 6 …etc.

# 4. Programming with comedi

## 4.1 Relationship between user program and device driver



### 4.2 Call flow of user's program




### 4.3 Subdevice

In comedi, the different functions of device are imaged to subdevice. So the varies functions of the hardware are show as subdevices, such as DI DO AI AO are all treated as different subdevices. Note that COMEDI_SUBD_FAKE is a special type of subdevice, used in digital input interrupt handling.

| Subdevice type | Subdevice series number | Description |
|---|---|---|
| COMEDI_SUBD_AI | 0 | Analog Input |
| COMEDI_SUBD_AO | 1 | Analog output |
| COMEDI_SUBD_DI | 2 | Digital input |
| COMEDI_SUBD_DO | 3 | Digital output |
| COMEDI_SUBD_DIO | 4 | Digital input and output |

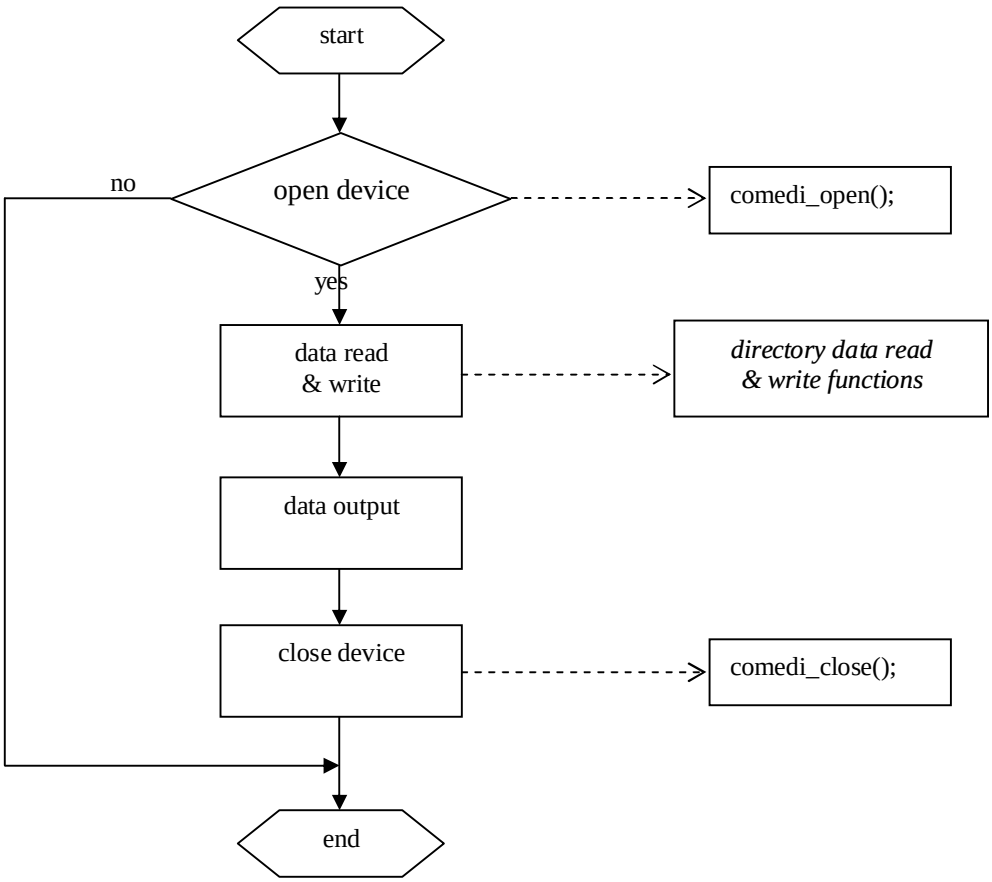
| | | |
|---|---|---|
| COMEDI_SUBD_COUNTER | 5 | Counter |
| COMEDI_SUBD_TIMER | 6 | timer |
| COMEDI_SUBD_MEMORY | 7 | Memory, EEPROM, DPRAM |
| COMEDI_SUBD_CALIB | 8 | Calibration DAC |
| COMEDI_SUBD_PROC | 9 | DSP |
| COMEDI_SUBD_UNUSED | 10 | Unused |
| COMEDI_SUBD_FAKE | 15 | FAKE device |

## 4.4 Subdevice operator mode

| MODE | FEATURE |
|---|---|
| Simple read & write mode | easy for program, but only can get one sample from one call |
| trigger mode | has 5 modes, one mode image to a device work mode |
| Command mode | use command ioctl, allows us to specify what causes each action that is taken during acquisition. |
| insn mode | through different function to realize different control |

### *4.4.1 Directory data read and write mode:*

This is the easiest way to control a device, comedilib has a common function to realize this operation . These function are: comedi_data_read(), comedi_data_write(), comedi_dio_read(), comedi_dio_write(), etc. The program steps are illustrated in the follow figure:



This is an example illustrate how to use this mode:

```
#include <stdio.h>
#include <comedilib.h>

int subdev = 0;                 /* data input subdevice */
int chan = 0;                   /* data input channel */
int range = 0;                  /* channel range */
int aref = AREF_GROUND;         /* channel aref */

int main(int argc,char *argv[])
{
        comedi_t *it;
        int chan=0;
        lsampl_t data;
        it=comedi_open("/dev/comedi0");
        comedi_data_read(it,subdev,chan,range,aref,&data);
        printf("%d\n",data);
        return 0;
}
```

### *4.4.2 Trigger mode*

***This is an old way, and will be removed from comedi in the later versions. It is suggested to use command mode rather than this mode.***

Most of our drivers support Trigger mode. Since it is an old mode, by default, drivers comment this part off by undefining some macro variables.

#undef CONFIG_COMEDI_MODE0

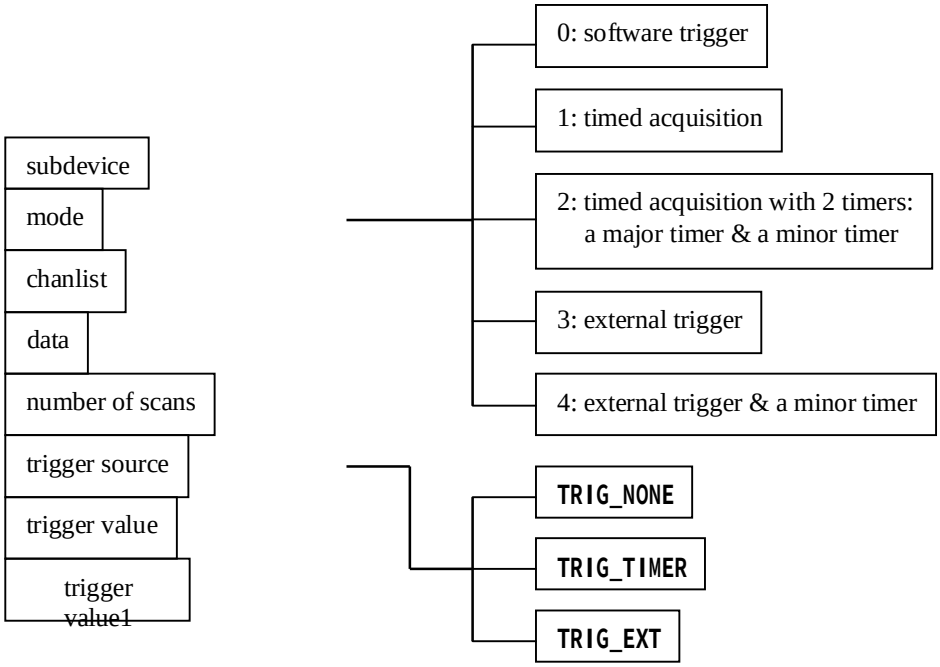
To enable trigger mode, just rebuild the driver by defining that macro variable.

#define CONFIG_COMEDI_MODE0

#### 4.4.2.1 comedi trigger structure

```
struct comedi_trig_struct{
        unsigned int subdev;            /* subdevice */
        unsigned int mode;              /* mode, include mode0,mode1…mode4*/
        unsigned int flags;
                TRIG_BOGUS              /* do the motions */
TRIG_DITHER             /* enable dithering */
```

```
TRIG_DEGLITCH                   /* enable deglitching */
TRIG_RT                         /* perform op in real time */
TRIG_CONFIG             /* perform configuration, not triggering */
TRIG_WAKE_EOS                   /* wake up on end-of-scan events */
TRIG_WRITE                      /* write to bidirectional devices */
        unsigned int n_chan;            /* number of channels */
        unsigned int *chanlist;
/* channel/range list */
/* the channel list determined which channels are sampled.
          In general, chanlist_len is the same as scan_end_arg.  Most
          boards require this.  */
        sampl_t *data;          /* data list, size depends on subd flags */
        unsigned int n;         /* number of scans */
        unsigned int trigsrc;
TRIG_ANY
TRIG_NONE               /* never trigger */
TRIG_NOW                /* trigger now + N ns */
TRIG_FOLLOW             /* trigger on next lower level trig */
TRIG_TIME               /* trigger at time N ns */
TRIG_TIMER              /* trigger at rate N ns */
TRIG_COUNT  /* trigger when count reaches N */
TRIG_EXT                /* trigger on external signal N */
TRIG_INT                /* trigger on comedi-internal signal N */
        unsigned int trigvar;           /* major timer value */
        unsigned int trigvar1;  /* minor timer value */
unsigned int data_len;  /* data list, size depends on subd flags */
        unsigned int unused[3]; /* unused */
};
```




#### 4.4.2.2 comparison between 5 modes

##### 4.4.2.2.1 mode 0

0 is one-shot. Actually, some of the drivers will return as many samples as you ask (i.e., trig.n), for the channels you specify. Others will limit it to a reasonable number that will finish in 1-10 ms, and the rest only return 1, for the first channel specified. The idea here is "at the convenience of the driver, preferably fast."

##### 4.4.2.2.2 mode1

mode 1 is timed acquisition, with a single timer. For example, if you request channels 2, 4, and 6, in mode 1, with the number of samples (trig.n) = 2, and a timer value that corresponds to 100 us, the driver programs the device to measure inputs at these times:

| time | chan |
|---|---|
| +0 us | 2 |
| +100 us | 4 |
| +200 | 6 |
| +300 | 2 |
| +400 | 4 |
| +500 | 6 |

If you board can support mode 2, mode 1 is not necessary. Comedilib will eventually emulate it.

#### 4.4.2.2.3 mode 2

mode 2 is similar, but has 2 timers, a major timer (trig.timeval) and a minor timer (trig.timeval1). Taking the previous example, but with a major timer of 1000 us and a minor timer of 100 us, inputs are measured at these times:

| time | chan |
| --- | --- |
| +0 us | 2 |
| +100 us | 4 |
| +200 | 6 |
| +1000 | 2 |
| +1100 | 4 |
| +1200 | 6 |

#### 4.4.2.2.4 mode 3

mode 3 is like mode 1, except that an external trigger is used. If t(n) represents the time of the nth trigger, inputs are measured at the following times:

| time | chan |
| --- | --- |
| t(1) | 2 |
| t(2) | 4 |
| t(3) | 6 |
| t(4) | 2 |
| t(5) | 4 |
| t(6) | 6 |

#### 4.4.2.2.5 mode 4

mode 4 uses an external trigger and a minor timer. With a minor timer of 100 us, we get:

| time | chan |
| --- | --- |
| t(1) | 2 |
| t(1)+100 us | 4 |
| t(1)+200 us | 6 |
| t(2) | 2 |
| t(2)+100 us | 4 |
| t(2)+200 us | 6 |

As a bit of background, mode 2 & 4 are the most useful. Mode 1 is the only supported by many boards. Mode 3 is the only supported by a few boards. Mode 3 could be useful to synchronize multiple boards, if the master board is capable of exporting a trigger signal.

### 4.4.2.3 Trigger mode example

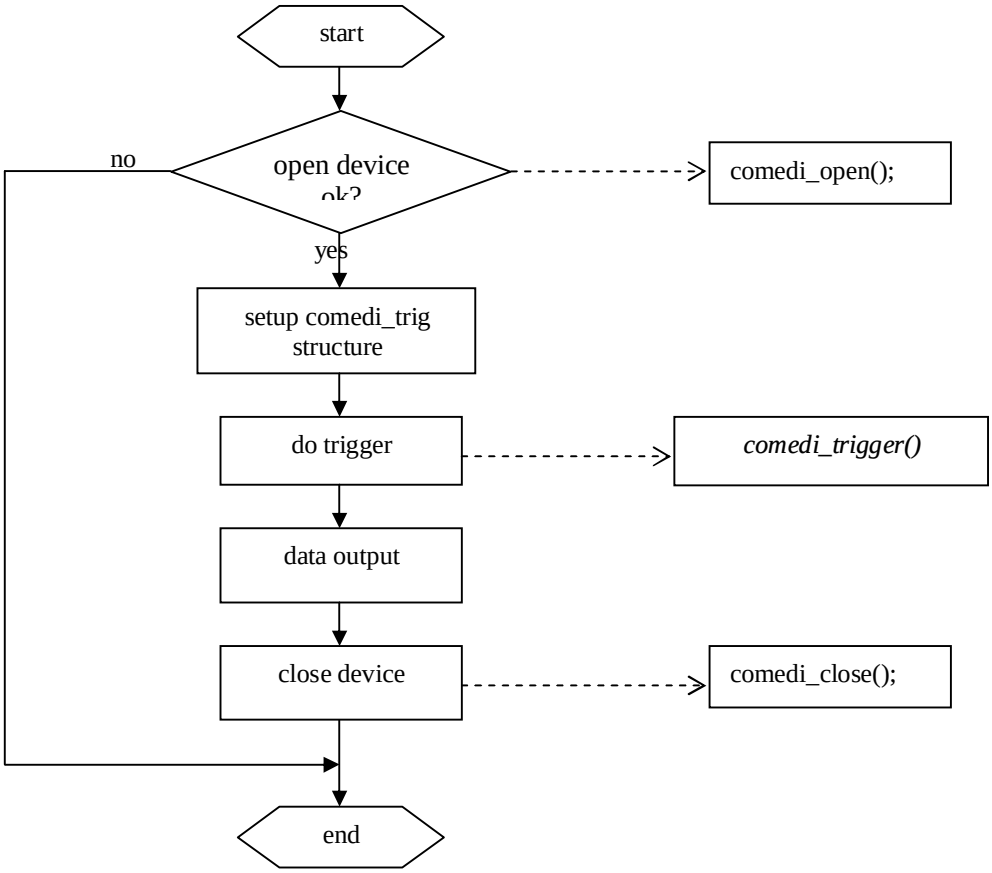
```
#include <stdio.h>
```

```
#include <comedilib.h>


char *filename="/dev/comedi0";
int verbose_flag;
comedi_t *device;
int subdevice = 0;
int channel = 0;
int aref;
int range = 0;
void main(int argc, char *argv[])
{
comedi_trig it;
    unsigned int chanspec;
    int save_errno;
    int ret;
    device = comedi_open(filename);
    if(!device){
        printf("E: comedi_open(\"%s\"): %s\n",filename,strerror(errno));
    }
    memset(&it,0,sizeof(it));
    it.subdev = subdevice;                  /* set the subdevice */
    it.mode = 0;                            /* set mode, in mode 0 */
    it.n_chan = 1;                          /* the length of channel list */
    it.chanlist = &chanspec;                /* set the channel list */
    it.data = (sampl_t )&data;              /* the acquisition data */
    it.n = 1;                               /* number of the acquisition data */
   chanspec = CR_PACK(channel, range, 0);   /* setup the channel */
    ret = comedi_trigger(device,&it);       /* do trigger */
    save_errno = errno;
    if(ret<0){
        printf("W: comedi_trig_ioctl: errno=%d %s\n",save_errno,
            strerror(save_errno));
    }
```

```
        printf("data: %d\n", data);
}
```

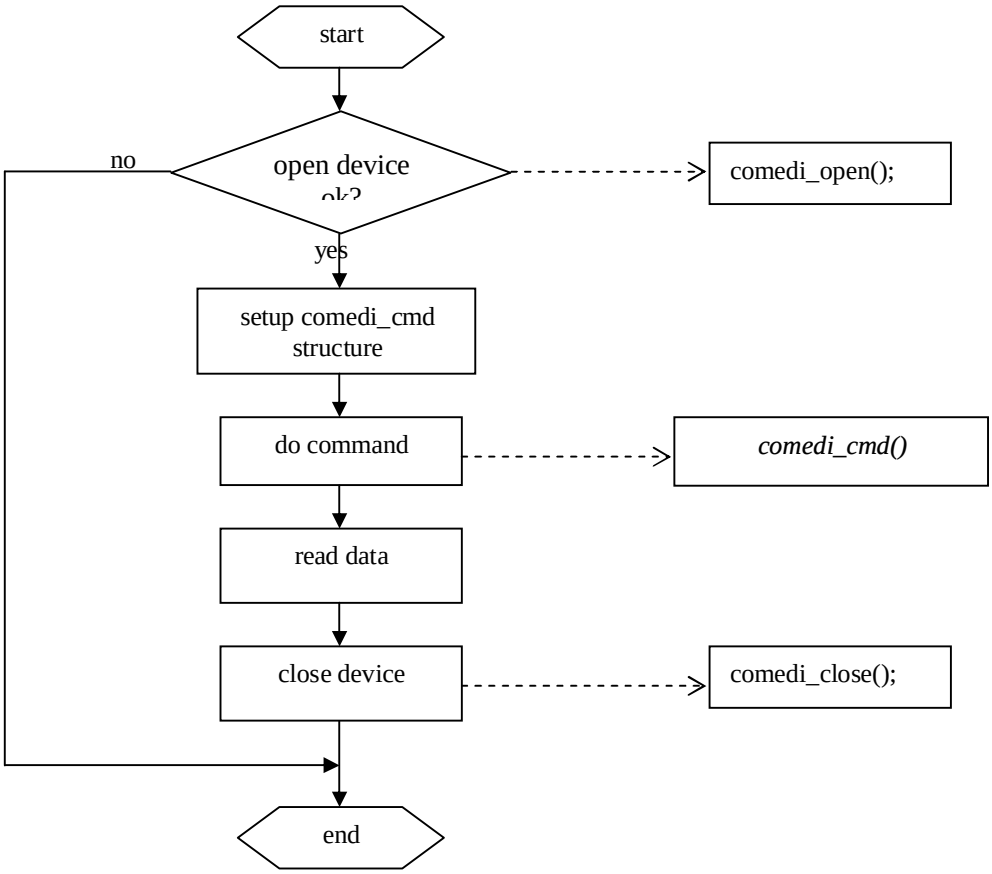
#### 4.4.2.4 Trigger mode usage



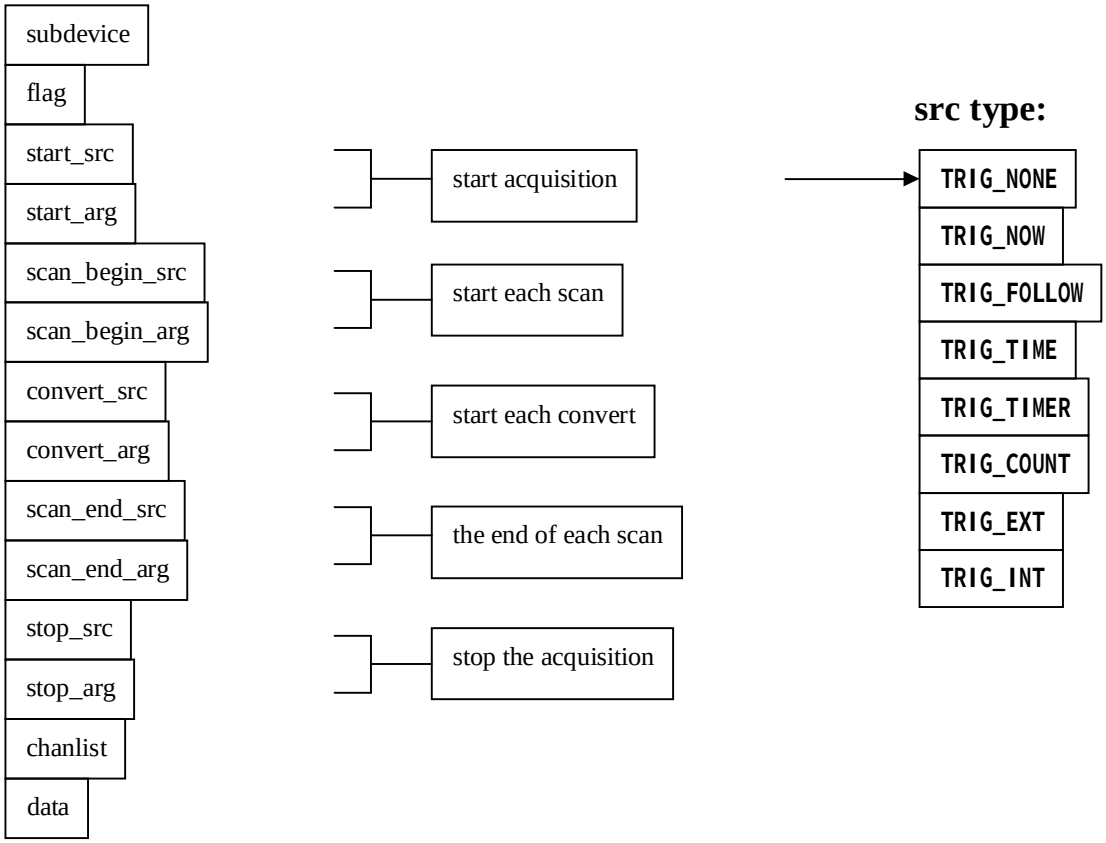
### *4.4.3 Command mode*

Command mode is the most powerful mode. The command structure includes the command structure to control the device, and then the comedilib function would execute this command.

With command structure, we can setup data IO trigger source, conversion start and end time, etc. After we set this parameters in the command structure, comedilib would pass these parameters and execute command as we wished.



#### 4.4.3.1 Command mode structure




```
struct comedi_cmd_struct{
unsigned int subdev;
        /* the subdevice that the command is sent to */

        unsigned int flags;
/* flags */
/* Wake up at the end of every scan */
        //cmd->flags |= TRIG_WAKE_EOS;
        /* Use a real-time interrupt, if available */
        //cmd->flags |= TRIG_RT;

        /* each event requires a trigger, which is specified
          by a source and an argument.  For example, to specify
```

```
            an external digital line 3 as a source, you would use
            src=TRIG_EXT and arg=3. */
          unsigned int start_src;
/* The start of acquisition is controlled by start_src.
          * TRIG_NOW:     The start_src event occurs start_arg nanoseconds
          *               after comedi_command() is called.  Currently,
          *               only start_arg=0 is supported.
          * TRIG_FOLLOW:  (For an output device.)  The start_src event occurs
          *               when data is written to the buffer.
          * TRIG_EXT:     start event occurs when an external trigger
          *               signal occurs, e.g., a rising edge of a digital
          *               line.  start_arg chooses the particular digital
          *               line.
          * TRIG_INT:     start event occurs on a Comedi internal signal,
          *               which is typically caused by an INSN_TRIG
          *               instruction.
          */
          unsigned int start_arg;
          /* argument */

          unsigned int scan_begin_src;
/* The start of acquisition is controlled by start_src.
          * TRIG_NOW:     The start_src event occurs start_arg nanoseconds
          *               after comedi_command() is called.  Currently,
          *               only start_arg=0 is supported.
          * TRIG_FOLLOW:  (For an output device.)  The start_src event occurs
          *               when data is written to the buffer.
          * TRIG_EXT:     start event occurs when an external trigger
          *               signal occurs, e.g., a rising edge of a digital
          *               line.  start_arg chooses the particular digital
          *               line.
          * TRIG_INT:     start event occurs on a Comedi internal signal,
          *               which is typically caused by an INSN_TRIG
          *               instruction.
```

```
        */

        unsigned int scan_begin_arg;
        /*in ns*/

        unsigned int convert_src;
/* The timing between each sample in a scan is controlled by convert.
        * TRIG_TIMER:   Conversion events occur periodically.
        *       The time between convert events is
        *       convert_arg nanoseconds.
        * TRIG_EXT:     Conversion events occur when an external trigger
        *       signal occurs, e.g., a rising edge of a digital
        *       line.  convert_arg chooses the particular digital
        *       line.
        * TRIG_NOW:     All conversion events in a scan occur simultaneously.
        * Even though it is invalid, we specify 1 ns here.  It will be
adjusted later to a valid value by comedi_command_test() */

        unsigned int convert_arg;
        /*in ns*/

        unsigned int scan_end_src;
/* The timing between each sample in a scan is controlled by convert.
        * TRIG_TIMER:   Conversion events occur periodically.
        *       The time between convert events is
        *       convert_arg nanoseconds.
        * TRIG_EXT:     Conversion events occur when an external trigger
        *       signal occurs, e.g., a rising edge of a digital
        *       line.  convert_arg chooses the particular digital
        *       line.
        * TRIG_NOW:     All conversion events in a scan occur simultaneously.
        * Even though it is invalid, we specify 1 ns here.  It will be
        * adjusted later to a valid value by comedi_command_test() */
```

```
        unsigned int scan_end_arg;
        /* number of channels */

        unsigned int stop_src;
/* The end of acquisition is controlled by stop_src and
        * stop_arg.
        * TRIG_COUNT:  stop acquisition after stop_arg scans.
        * TRIG_NONE:   continuous acquisition, until stopped using
        *          comedi_cancel()
        **/
        unsigned int stop_arg;
        /* number of channels */

        unsigned int *chanlist;
/* channel/range list */
/* the channel list determined which channels are sampled.
          In general, chanlist_len is the same as scan_end_arg.  Most
          boards require this.  */
        unsigned int chanlist_len;

        sampl_t *data;
/* data list, size depends on subd flags */
        unsigned int data_len;
        /*length of data*/
};
```

#### 4.4.3.2 command example

```
#include <stdio.h>
#include <comedilib.h>

#define N_SCANS             10
#define N_CHANS             16

char *filename="/dev/comedi0";
int subdevice = 0;
```

```
int chan=0;
int range = 0;
int aref = AREF_GROUND;
int n_chan = 1;

#define BUFSZ 100
char buf[BUFSZ];
sampl_t capdata[50];

unsigned int chanlist[N_CHANS];

void do_cmd(comedi_t *dev,comedi_cmd *cmd);

int main(int argc, char *argv[])
{
        comedi_t *dev;
        comedi_cmd cmd;
        dev = comedi_open(filename);
        if(!dev){
                perror(filename);
                exit(1);
        }
        memset(cmd,0,sizeof(cmd));
        cmd->subdev = subdevice;
        cmd.flags =     0;
/* each event requires a trigger, which is specified by a source and an argument.  For example, to specify an external digital line 3 as a source, you would use src=TRIG_EXT and arg=3. */
/* In this case, we specify using TRIG_NOW to start acquisition immediately when the command is issued.  The argument of TRIG_NOW is "number of nsec after NOW", but no driver supports it yet.  Also, no driver currently supports using a start_src other than TRIG_NOW. */

        cmd->start_src =                TRIG_NOW;
        cmd->start_arg =                0;
```

```
/* The timing of the beginning of each scan is controlled by scan_begin.  TRIG_TIMER
specifies that scan_start events occur periodically at a rate of scan_begin_arg nanoseconds
between scans. */


        cmd->scan_begin_src =  TRIG_FOLLOW;
        cmd->scan_begin_arg = 0;


/* The timing between each sample in a scan is controlled by convert.  Like above,
TRIG_TIMER specifies that convert events occur periodically at a rate of convert_arg
nanoseconds between scans. */


        cmd->convert_src =      TRIG_TIMER;
        cmd->convert_arg =      100000;


/* The end of each scan is almost always specified using TRIG_COUNT, with the argument
being the same as the number of channels in the chanlist.  You could probably find a device
that allows something else, but it would be strange. */


        cmd->scan_end_src =     TRIG_COUNT;
        cmd->scan_end_arg =     n_chan;          /* number of channels */


/* The end of acquisition is controlled by stop_src and stop_arg.  The src will typically be
TRIG_COUNT or TRIG_NONE.  Specifying TRIG_COUNT will stop acquisition after
stop_arg number of scans, or TRIG_NONE will cause acquisition to continue until stopped
using comedi_cancel(). */


        cmd->stop_src =                 TRIG_COUNT;
        cmd->stop_arg =                 N_SCANS;


/* the channel list determined which channels are sampled.In general, chanlist_len is the
same as scan_end_arg.  Most boards require this.  */


        cmd->chanlist =         chanlist;
        cmd->chanlist_len =     n_chan;


        cmd->data = capdata;
        cmd->data_len = 10*sizeof(sampl_t);
```

```
	chanlist[0]=CR_PACK(chan+0,range,aref);
	chanlist[1]=CR_PACK(chan+1,range,aref);
	chanlist[2]=CR_PACK(chan+2,range,aref);
	chanlist[3]=CR_PACK(chan+3,range,aref);

	do_cmd(dev,&cmd);

	return 0;
}
void do_cmd(comedi_t *dev,comedi_cmd *cmd)
{
	int total=0;
	int ret, i, data;
	int go;

	ret=comedi_command(dev,cmd);

	if(ret<0){
		comedi_perror("comedi_command");
		return;
	}

	go=1;
	while(go){
		ret=read(comedi_fileno(dev),buf,BUFSZ);
		if(ret<0){
			if(errno==EAGAIN){
				usleep(10000);
			}else{
				go = 0;
				perror("read");
			}
		}else if(ret==0){
```

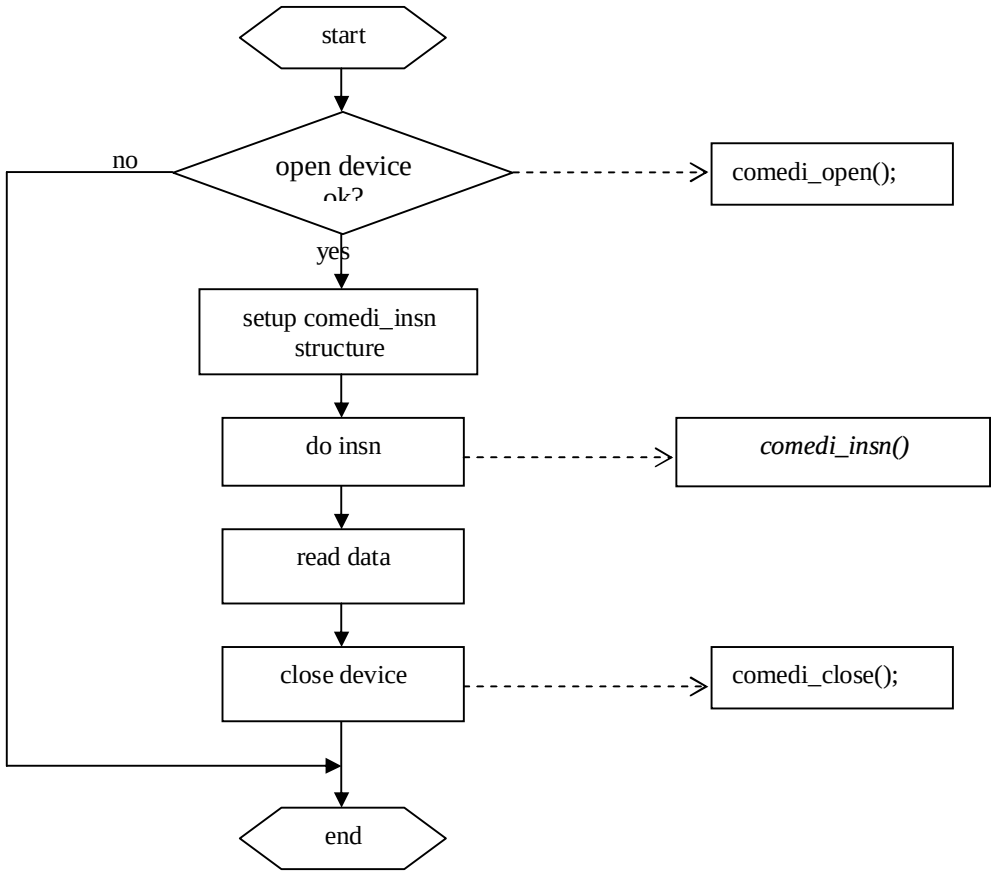
```
                        go = 0;
                }else{
                        static int col = 0;
                        int i;
                        total+=ret;
                        for(i=0;i<ret/2;i++){
                                printf("%d ",((sampl_t *)buf)[i]);
                                printf("%f  |  ",(comedi_to_phys((float)((sampl_t  *)buf)[i],
srange[col], maxdata[col])));
                                col++;
                                if(col==n_chan){
                                        printf("\n");
                                        col=0;
                                }
                        }
                }
        }
}
```

### *4.4.4 insn mode*

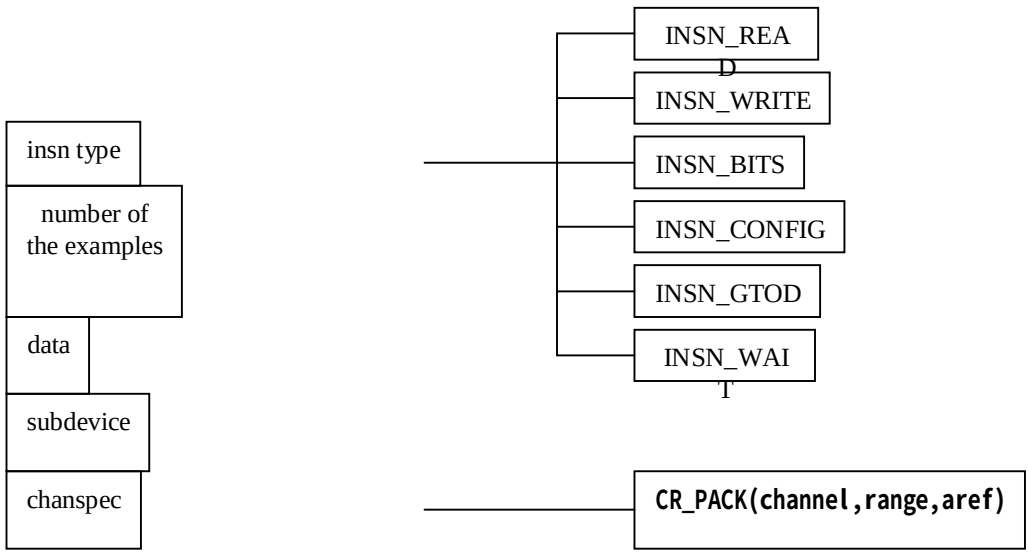
Insn mode can almost be used for every type of subdevice. When using insn mode, we need to setup insn type, so comedi could call different function with the different insn type. In device driver, we need to has different insn function referring to different insn type.

The insn structure also has some other data item we can use to do various operations. So the obviously difference between insn mode and command mode is different type insn call different function in the driver, while the command mode simply send a command to the driver.

#### 4.4.4.1 Insn mode usage




#### 4.4.4.2 insn structure




```
struct comedi_insn_struct{
unsigned int insn;
```

```
/*what would the insn do:
INSN_GTOD:          perform a gettimeofday(), see comedi_fops.c ln 589.
INSN_READ:          do analog input read;
INSN_WRITE:         output data;
INSN_BITS:          di/do bit operatoin;
INSN_CONFIG:                dio config the transfer direction;
INSN_WAIT:          delay for *data ns. See comdei_fops.c ln600.
*/
unsigned int n;
/*loop time of insn operation
        must < 256
*/

lsampl_t *data;
        /*data buffer pointer where the samples been put.
        */

unsigned int subdev;
/* the subdevice that the command is sent to */

        unsigned int chanspec;
        unsigned int unused[3];
};
```

#### 4.4.4.3 insn example

```
#include <comedilib.h>
#define N_SAMPLE 8
char *filename="/dev/comedi0";
comedi_t *device;

int subdevice = 0;
int channel = 0;
int aref;
int range = 0;
```

```
int main(int argc, char *argv[])
{
	lsampl_t data[N_SAMPLE];
	int save_errno;
	int ret, i;
comedi_insn insn[3];
	comedi_insnlist il;
	int n_channels, n_range;

	device = comedi_open(filename);
	if(!device){
		printf("E: comedi_open(\"%s\"): %s\n",filename,strerror(errno));
	}
memset(&il,0,sizeof(il));
	memset(insn,0,sizeof(insn));

	il.n_insns = 3;
	il.insns = insn;

	insn[0].insn = INSN_GTOD;
	insn[0].n=2;
	insn[0].data = (void *)&t1;

insn[1].subdev = subdevice;
	insn[1].insn = INSN_READ;
	insn[1].n = N_SAMPLE;
	insn[1].chanspec = CR_PACK(channel, range, 0);
	insn[1].data = &data;

	insn[2].insn = INSN_GTOD;
	insn[2].n=2;
	insn[2].data = (void *)&t2;

	ret = comedi_do_insnlist(device,&il);
```

```
save_errno = errno;


    if(ret<0){
        printf("W: comedi_do_insn: errno=%d %s\n",save_errno,strerror(save_errno));
    }


    for (i=0; i<N_SAMPLE; i++) {
        printf("%d     ", data[i]);
    }
printf("\nread time: %ld us\navr time:  %ld us\n",
(t2.tv_sec-t1.tv_sec)*1000000+(t2.tv_usec-t1.tv_usec),
 (t2.tv_sec-t1.tv_sec)*1000000+(t2.tv_usec-t1.tv_usec)/ N_SAMPLE);
    return 0;
}
```

#### 4.4.4.4 insn convention

| subdev | insn | data format | NOTE |
| --- | --- | --- | --- |
| 2 | INSN_BITS | data[0] - reserved<br>data[1] – DI value | |
| 3 | INSN_BITS | data[0] – channel mask<br>data[1] – DO value | For example, DO at channel 2, mask would be 0x4 and value should be 0x4 or 0xf. |
| 15 | INSN_BITS | data[0] – pid of user case<br>data[1] – action dealing with pid in data[0] | data[1]=0 - clear a pid from driver<br>data[1]=1 - append a pid to driver<br>data[1]=2 - clear all pid in driver |

#### 4.4.4.5 insn special hint

1. PCM-3730
   In INSN_BITS function of DI, the low 8 bits of data[1] correspond to the IDI bits 0-7, and the next high 16 bits correspond to the DI bits 0-16.
   In INSN_BITS function of DO, the low 8 bits of data[1] correspond to the IDO bits 0-7, and the next high 16 bits correspond to the DO bits 0-16.

### 4.5 Advantech test example manuals

Advantech provides test example for each DAQ cards as well as manuals, which located at the document directory of test package :

**test/Doc/<card id>/manual.pdf**

Please refer to these manuals for usage of each cases.

## 5. comedilib functions description

### 5.1 Device file operation function

- **comedi_t *comedi_open(char *fn);**

  Opens a comedi device specified by the filename fn.

Argument:

**fn** is a device filename, such as "/dev/comedi0";

Returns: NULL on error. On sucess, it returns a handle that is given as a parameter to other libcomedi functions.

- **void comedi_close(comedi_t *it);**

Closes a device previously opened by comedi_open().

Argument:

it is the values return from comedi_open().

- **int comedi_fileno(comedi_t *it);**

This function returns the integer descriptor for the handle it. It is equivalent to the standard function fileno.

Argument:

**it** is the values return from comedi_open().

Return: If it is an invalid comedi_t pointer, the function returns -1 and sets the appropriate libcomedi error value.

## 5.2 Subdevice operation function

- **int comedi_get_n_subdevices(comedi_t *it);**

This function returns the number of subdevices associated with the comedi descriptor it.

Argument:

**it** is the values return from comedi_open().

Returns: Argument: it is the values return from comedi_open(), or -1 if there is an error.

- **int comedi_find_subdevice_by_type(comedi_t *it,int type,unsigned int start_subdevice);**

This function tries to locate a subdevice belonging to comedi device it.

Argument:

**type** is one of subdevice type, include:

- COMEDI_SUBD_UNUSED Subdevice has no functionality.
- COMEDI_SUBD_AI Analog input
- COMEDI_SUBD_AO Analog output
- COMEDI_SUBD_DI Digital input
- COMEDI_SUBD_DO Digital output
- COMEDI_SUBD_DIO Digital input/output.
- COMEDI_SUBD_COUNTER Counter
- COMEDI_SUBD_TIMER Timer
- COMEDI_SUBD_MEMORY Memory, e.g., EEPROM or dual-ported RAM

- COMEDI_SUBD_CALIB Calibration DACs
- COMEDI_SUBD_PROC Processor or DSP

**start_subdevice** define the starting subdevice number.

Returns: If it finds the requested subdevice, it returns its index. If it does not locate the requested subdevice, it returns -1 and sets the comedi error number to "subdevice not found". If there is an error, the function returns -1 and sets the appropriate error.

### 5.3 Driver information acquirement

Ø **char *comedi_get_driver_name(comedi_t *it);**

This function returns the name of a driver.

Argument:

**it** is the values return from comedi_open().

Returns: a pointer to a string containing the name of the driver being used by comedi for the comedi device represented by it. This pointer is valid until the comedi descriptor it is closed, returns NULL if there is an error.

Ø **char *comedi_get_board_name(comedi_t *it);**

This function returns the name of the device.

Argument:

**it** is the values return from comedi_open().

Return: a pointer to a string containing the name of the device. This pointer is valid until the comedi descriptor it is closed. returns NULL if there is an error.

Ø **int comedi_get_subdevice_type(comedi_t *it,unsigned int subdevice);**

This function get an integer describing the type of subdevice that belongs to the comedi device.

Argument:

**it** is the values return from comedi_open().

**subdevice** is the index of the subdevice.

Returns: the type of this subdevice. -1 is returned, if is there is an error.

Ø **int comedi_get_n_channels(comedi_t *it,unsigned int subdevice);**

This function gets the number of channels of a subdevice.

Argument:

**it** is the values return from comedi_open().

**subdevice** is the index of the subdevice.

Returns: the number of a subdevice channels. -1 is returned, if is there is an error.

Ø **lsampl_t comedi_get_maxdata(comedi_t *it,unsigned int subdevice,unsigned int chan);**

This function gets the maximum valid data value for channel of a subdevice.

Argument:

**it** is the values return from comedi_open().

**subdevice** is the index of the subdevice,

**chan** is the index of the channel.

Returns: the maximum valid data value of the channel or returns 0 on error.

- **comedi_range * comedi_get_range(comedi_t *it,unsigned int subdevice,unsigned int chan,unsigned int range);**

This function gets the range for channel of a subdevice.

Argument:

**it** is the values return from comedi_open().

**subdevice** is the index of the subdevice,

**chan** is the index of the channel,

**range** is the range index of a subdevice.

Returns: a pointer to a comedi_range structure that contains information that can be used to convert sample values to or from physical units. The pointer is valid until the comedi device it is closed. If there is an error, NULL is returned.

## 5.4 Data acquisition funcion

- **int comedi_data_read(comedi_t *it,unsigned int subd,unsigned int chan, unsigned**

int range,unsigned int aref,lsampl_t *data);

This function Reads a single sample from the subdevice.

Argument:

**it** is the values return from comedi_open().

**subd** is the index of the subdevice,

**chan** is the index of the channel,

**range** is the range index of a subdevice,

**aref** is analog reference type, (For the A/D conversion (if appropriate), the device is configured to use range specification range and (if appropriate) analog reference type aref. Analog reference types that are not supported by the device are silently ignored.) data contain the result data.

Returns: On sucess, comedi_data_read() returns 0. If there is an error, -1 is returned.

- **int comedi_data_write(comedi_t *it,unsigned int subd,unsigned int chan, unsigned int range,unsigned int aref,lsampl_t data);**

This function writes a single sample on the channel of a subdevice.

Argument:

**it** is the values return from comedi_open().

**subd** is the index of the subdevice,

**chan** is the index of the channel,

**range** is the range index of a subdevice,

**aref** is analog reference type,

**data** contain the result data.

Returns: on sucess, comedi_data_write() returns 0. If there is an error, -1 is returned.

- **int comedi_dio_read(comedi_t *it,unsigned int subd,unsigned int chan,unsigned int *bit);**

  This function reads the status of channel belonging to the digital input subdevice.

  Argument:

  **it** is the values return from comedi_open().

  **subd** is the index of the subdevice,

  **chan** is the index of the channel,

  **bit** contains the result, 0 or 1.

  Returns: return 0 on success, returns -1 on failure.

  This function is equivalent to comedi_data_read(it,subd,chan,0,0,bit).

- **int comedi_dio_write(comedi_t *it,unsigned int subd,unsigned int chan, unsigned int *bit);**

  This function reads the status of channel belonging to the digital input subdevice.

  Argument:

  **it** is the values return from comedi_open().

  **subd** is the index the the subdevice,

  **chan** is the index of the channel belonging to the digital output device,

  **bit** contains the data that will be write to the channel, 0 or 1.

  Returns: return 0 on success, returns -1 on failure.

- **int comedi_dio_bitfield(comedi_t *it,unsigned int subd,unsigned int write_mask,unsigned int *bits);**

  This function allows multiple channels to be read simultaneously from a digital input or digital I/O device.

  Argument:

  **it** is the values return from comedi_open().

  **subd** is the index of the subdevice,

  **write_mask** is the mask of a channel,

  **bit** contains the data that will be write to the channel, 0 or 1.

  Returns: return 0 on success, returns -1 on failure.

- **comedi_trigger(comedi_t *it, comedi_trig *trig);**

  This function instructs comedi to perform the command specified by the trigger structure.

  Argument:

  **it** is the values return from comedi_open(),

  **trig** is the comedi_trig structure, as we motioned before,

  Results: depend on the particular command being issued. If there is an error, -1 is returned.

- **comedi_command(comedi_t *it, comedi_cmd *cmd);**

This function instructs comedi to perform the command specified by the command structure.

Argument:

**it** is the values return from comedi_open(),

**cmd** is the comedi_cmd structure, as we motioned before

Results: depend on the particular command being issued. If there is an error, -1 is returned.

- **comedi_do_insn(comedi_t *it, comedi_insn *insn);**

This function instructs comedi to perform the command specified by the insn structure.

Argument:

**it** is the values return from comedi_open(),

**insn** is the comedi_insn structure, as we motioned before

Results: depend on the particular command being issued. If there is an error, -1 is returned.