Data Acquisition with Comedi

One standard platform provides a uniform API for many data acquisition boards.

by Caleb Tennis

Most scientists and engineers love data. The more data you can feed them, the more they smile. In a laboratory setting, data means everything. In order to spot trends, analyze strange phenomena and draw final conclusions, a lab person needs to make sure they have acquired a complete set of data.

The concept of data acquisition therefore encompasses a broad scope of ideas. Most scientists and engineers, however, agree that data acquisition is the result of the measurement of some natural process. This could be as simple as the measurement of a temperature, for example, or as complex as the measurement of impurities in molten steel.

In the computing world, data acquisition most commonly is done by measuring a voltage. To do so, it is necessary to have some sensor or measurement device that is capable of producing a voltage that the computer can measure. It's also important to know the correlation between the measured parameter and the sensor's voltage output. Ideally, the correlation is linear, as in a temperature sensor where 1 measured degree Celsius corresponds to .1 volts.

Modern motherboards have onboard sensors, such as National Semiconductor's LM78, which assess the overall health of the system. These sensors measure such conditions as cooling fan speeds, processor core voltages and temperatures and hard drive rotation speeds. This information is acquired by the chip and can be reported to the processor through a serial bus. The open-source project lm_sensors (secure.netroedge.com/~lm78) provides the software for monitoring many aspects of motherboards.

Typical personal computers have no common interface for analog data acquisition, however. In order to make some external voltage measurement, a new interface is necessary. Data acquisition (DAQ) cards designed for either the PCI or ISA bus fill this gap. Many manufacturers make cards well suited for taking external measurements.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Inputs</td>
<td>Measure external signals, such as a voltage</td>
</tr>
<tr>
<td>Analog Outputs</td>
<td>Send a variable signal</td>
</tr>
<tr>
<td>Digital Inputs/Outputs</td>
<td>A discrete on/off signal; commonly 0 for off, 5 volts for on</td>
</tr>
<tr>
<td>Counters</td>
<td>Can count a number of pulses or measure frequency</td>
</tr>
<tr>
<td>Timers</td>
<td>Can measure the amount of time elapsed between two digital pulses</td>
</tr>
</tbody>
</table>

The Comedi Project

Most Linux users have experienced firsthand the complications surrounding having a single type of system (a
printer, for example) and multiple models, makes, vendors and drivers. Any attempt at standardization
becomes a large project. If the project receives enough support, it becomes the standard. Some vendors, like
National Instruments, have released Linux drivers for their DAQ products, while others have not.

Comedi, or Control and Measurement Device Interface, is the standard suite of data acquisition drivers and
libraries for Linux. Started in 1996 by David Schleef, Comedi attempts to support multiple vendors and
models of cards through a common interface. In fact, the overall API design is a balance between modularity
and complexity. Like other Linux driver projects, some of the work is the result of a lot of reading of
hardware manuals, some is the result of reverse engineering and some is the result of manufacturers' assistance in providing Comedi support for their products.

How It Works

Comedi is separated into two parts. Comedi itself is the package of drivers that are loaded into kernel space,
and comedilib gives user-space access to those drivers. It is through comedilib that the transparency of
Comedi shines. Programs using Comedi can be written in C or C++. Perl and Python bindings also exist for
Comedi.

Comedi breaks things down into channels, subdevices and devices. A channel is the lowest level of
measurement or control. Multiple channels of the same type are grouped into a common set, called a
subdevice. Then multiple subdevices are grouped together into a complete device. When using Comedi, first
a Comedi driver is loaded into memory. Then, /usr/sbin/comedi_config is run to bind the driver to a
Comedi device, such as /dev/comedi0. Finally, functions are available in comedilib to access the various
devices on the DAQ card.

A Lab Example

One example of an application for DAQ and Comedi is the Analytical Engineering, Inc. (AEI) airflow
laboratory. In the AEI lab, airflow is generated by a fan and is forced through orifices of varying sizes. Using
a custom-written software application, a technician can monitor the pressure buildup across the orifice. In
turn, this pressure buildup can be used to calculate the approximate amount of air flowing across the orifice.
This calculation is vital, because it allows a technician to determine whether various meter calibrations are
correct.

However, the actual mass flow is more difficult to calculate completely. This number requires knowledge of
two air pressures, three airflow temperatures, humidity, barometric pressure and altitude.

Off-the-shelf components exist for converting these measurements to voltage; one of the most popular
interfaces is 5B. Using 5B modular blocks, it's possible to transform all of these measurements to voltages
the DAQ card can read.
Using Comedi, reading these voltages becomes as trivial as using the \texttt{comedi\_data\_read} function. Calling this function and specifying a certain channel produces a resultant value, 3,421 for instance. But what does this number mean?

DAQ cards measure with a certain bit precision, 12 bits being the most common. They also specify a range or ranges of voltages over which they can be programmed to measure. Because a 12-bit number is represented from 0 to 4,095, it's easy to see that 3,421 is simply $3,421/4,095 \times 100\%$ of full scale (4,095). If the range of voltages is specified as [0, 5], then 3,421 would represent 4.177 volts.

Utilizing this information and knowing that the 5B block for temperature maps as [0 volts – 5 volts] → [0°C – 100°C], a small amount of programmatic math delivers a temperature of 83.56°C. Couple all of these measurements together, add a nice GUI interface and repeat the DAQ process every second.

More complex data acquisition can be performed as well. When acquiring data, it's important to make sure you sample fast enough so as not to miss any important information that occurs between samples. To support this, Comedi offers a command interface that can be used to set up synchronized sampling. Based on the sophistication of the DAQ card, timing can be handled by software interrupts or on-card interrupts.

\textbf{Listing 1. Sample Program for Acquiring Voltage from One Channel}

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

const char *filename = "/dev/comedi0";
int main(int argc, char *argv[])
{
  lsampl_t data;
  int ret;
  comedi_t *device;

  /* Which device on the card do we want to use? */
  int subdevice = 0;
  /* Which channel to use */
  int channel = 0;
  /* Which of the available ranges to use */
  int range = 0;
  /* Measure with a ground reference */
  int analogref = AREF_GROUND;

  device = comedi_open(filename);
```

Figure 1. Airflow Measurement Device
if(!device){
  /* We couldn't open the device - error out */
  comedi_perror(filename);
  exit(0);
}

/* Read in a data value */
ret=comedi_data_read(device,subdevice,
    channel,Range,analogref,&data);

if(ret<0){
    /* Some error happened */
    comedi_perror(filename);
    exit(0);
}

printf("Got a data value: %d\n", data);
return 0;
}

Comedi shines in most data acquisition applications. In fact, Comedi's limit generally resides in the hardware on which it's being run. Less expensive cards typically have a slower scan rate ability. For fast data acquisition, most of the higher priced cards come with onboard DMA, allowing an onboard processor to handle the acquisition and allowing Comedi simply to route the acquired buffered data.

Listing 2. Code Snippet Demonstrating More Advanced Scanning by Using Commands and Triggers

/* Goal: Set up Comedi to acquire 2 channels, and
    scan each set twice. Perform the acquisition
    after receiving a trigger signal on a digital line.
    */
comedi_cmd c, *cmd=&c;
unsigned int chanlist[2];

/* CR_PACK is a special Comedi macro used to
    setup a channel, a range, and a ground reference */
chanlist[0] = CR_PACK(0,0,0);
chanlist[1] = CR_PACK(1,0,0);

/* Which subdevice should be used? */
/* Subdevice 0 is analog input on most boards */
cmd->subdev = 0;
cmd->chanlist = chanlist;
cmd->chanlist_len = n_chan;

/* Start command when an external digital line
    is triggered. Use digital channel specified
    in start_arg */

cmd->start_src = TRIG_EXT;
cmd->start_arg = 3;

/* begin scan immediately following trigger */
/* cmd->scan_begin_src = TRIG_FOLLOW; */
cmd->scan_begin_arg = 0;

/* begin conversion immediately following scan */
cmd->convert_src = TRIG_NOW;

/* end scan after acquiring scan_end_arg channels */
  cmd->scan_end_src = TRIG_COUNT;
  cmd->scan_end_arg = 2;

/* Stop the command after stop_arg scans */
  cmd->stop_src = TRIG_COUNT;
  cmd->stop_arg = 2;

/* Start the command */
comedi_cmd(device, cmd);

Fast scan rates don't translate to fast processing, however. Due to the non-deterministic nature of the stock Linux kernel, it's virtually impossible to handle acquisition and processing in real time—that is, to maintain strict scheduling requirements for a process. Help is available, however. The Linux Real-Time Application Interface (RTAI) and RTLinux are two of a small number of add-on packages that allow for better timing control in the kernel. Both packages provide interfaces to Comedi.

The basic idea behind these real-time interfaces is simple. Instead of running the kernel as the monolithic process, run it as a child of a small and efficient scheduler. This design prevents the kernel from blocking interrupts and allows it to be preempted. Then, any application that needs real-time control of the system can register itself with the scheduler and preempt the kernel as often as it needs to.

Figure 2. Normal Linux Process Scheduling vs. Real-Time Linux Process Scheduling

A Lab Example

AEI maintains a number of testing chambers for diesel engines, known as test cells. In a cell, an engine is equipped with a number of temperature and pressure measurement devices. A frequency measurement device also is used to measure the rotational speed of the engine. Finally, the engine is connected to a dynamometer, which simulates actual driving conditions by varying the resistance against the spinning engine. This results in generated torque, which is measured as well.
Figure 3. An Engine Being Instrumented

The actual scan rate of the engine data is slow, only 20 times per second. If the measurement of this data were the only required job, the overall setup would be straightforward. However, a number of variable parameters must be tuned and controlled with the newest acquisition of each set of numbers. The engine throttle position and dynamometer load amounts must be varied slightly to maintain the engine speed at a specific condition. Valves in the cell controlling cooling water flow must be adjusted to keep engine coolant temperatures at constant levels. Safety measures must be checked to determine that no catastrophic problem has occurred.

Figure 4. Overview of Engine Measurement and Control with Comedi

All of these checks and new control values must be taken care of before the kernel can return to handling the rest of its scheduling. If the Linux kernel were to handle this scheduling on its own, it is quite possible that everything would work properly. However, it's impossible to determine beforehand when each stage of the process will be executed. With real-time extensions, however, the problem becomes trivial.

A real-time kernel is not without its downsides. While the real-time scheduler is executing some process at a fixed interval, the Linux kernel basically is put on hold. This means that a real-time process must be fast and efficient, and it must relinquish control back to the kernel as quickly as possible. Failure to do so results in sluggishness in the non-real-time portion of the system. If something goes wrong in the real-time process and control never goes back to the kernel, a complete system lockup can occur as well.
A Practical Example

Laboratory aside, sometimes it's interesting and fun to put Comedi to work at home. Low-end multipurpose data acquisition cards can be purchased for $99–$299 US, depending on brand, complexity and acquisition rate. Some examples of home projects include monitoring temperature in various parts of the house or scanning a magnetic sensor on a garage door to remind you that it's still open.

One interesting aspect of the personal computer is that parallel port lines can be controlled individually. Using Comedi, it's trivial to turn on and off these digital lines. When used with some form of relay, these digital lines can turn off and on anything imaginable.

Although parallel ports toggle between 0 and 5 volts, they typically do not have the capacity to source much electrical current. That said, it's a bad idea to connect the parallel port line directly to a device to turn it on or off without adding some kind of buffer circuitry. Many Web sites exist that explain how to create these circuits.

I use Comedi, an old 486 and two parallel ports to create an annual holiday light show. Lights are hung on the house in normal fashion, and a pair of wires for each set of lights is run back into the control room (a spare bedroom, in this instance). These power wires are connected to a custom-built circuit board that houses mechanical relays that send the power to the lights when they receive a 5-volt signal from the parallel port. A simple C program uses Comedi function calls to control the parallel port lines digitally, that is, to turn on and off the lights. Simple text files tell the program when to turn various lights on and off. And, the neighborhood receives a treat.

Figure 5. Interface Board for Parallel Port Light Show Display

Conclusion

Data acquisition is extremely valuable in the laboratory. The generic interface that Comedi provides allows great ease of use in Linux for a large number of available DAQ cards. As the popularity of Linux grows, the importance of having an interface such as Comedi's becomes vital.

Furthermore, as the low-end DAQ cards become even less expensive, Linux-based data acquisition becomes more and more appealing to hobbyists and do-it-yourselfers. What used to be an expensive set of software and hardware now is a viable method of implementation for a multitude of applications.

Resources for this article: www.linuxjournal.com/article/7610.

Caleb Tennis has been using Linux since 1996. He was the release coordinator of the KDevelop Project and
now is focusing his attention on maintaining KDE for Gentoo. Besides overseeing engineering at a diesel
engine test facility, he also teaches Linux part-time at a local college.