How to create a very-low cost, very-low power, credit-card sized and real-time ready datalogger

Maxime Bès de Berc (1), Marc Grunberg (2), Fabien Engels(2)

(1) CNRS UMR7516, Seismology Laboratory, EOST, France (mbesdeberc@unistra.fr), (2) CNRS UMS830, Seismology Laboratory, EOST, France

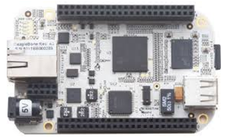
In some cases a field instrumentalist could have to add some extra sensors in a remote station (for environmental parameters by example). Additional ADCs (Analogic Digital Converters) are not always implemented on commercial dataloggers, or may already be used. Adding more ADCs often implies an expensive development, or buy a new datalogger. We present here a very simple way to deploy an embedded ARM computer, use its features and embedded ADCs to create datas in a seismological standard format and integrating it within the near-real-time seismological data stream from the station as a secondary source.

Our specifications: what do we need exactly?

Our goal was to be able to quickly deploy a system able to run in a standalone mode in a remote site for digitizing various parameters. As those parameters are very various, we could afford a little analogic conditioner for each. All datas and protocol have to be in a standard format for seismology (named SEED: Standard for the Exchange of Earthquake Data).

We listed our requirements:

Several embedded ADCs.
Easy way to setup and read the values
Standard OS
Cheap system
Synchronized through network, and possibility to synchronize with a GPS receiver
Datas with a sampling rate up to 100Hz
Able to buffer the datas
Able to save datas locally in mini-seed (seed without metadat) format
Able to send datas in near-real-time transmission in seedlink protocol (for sending mini-seed datas through internet)
Able to send older datas on demand with rsync



We quickly chose the Beaglebone (or BeagleBone Black) as it fits our hardware requirements. *"BeagleBone is an $90, credit-card-sized Linux computer that connects to the Internet and runs software such as Debian. With plenty of I/O and processing power for real-time analysis provided by an AM335x 720MHz ARM processor, BeagleBone can be complemented with cape plug-in boards to augment functionality."* (A) We bought a board and decided to use the well-known Seiscomp3 software toolkit, available on www.seiscomp3.org/wiki/download. *"SeisComP is a seismological software for data acquisition, processing, distribution and interactive analysis that has been developed by the GEOFON Program at Helmholtz Centre Potsdam, GFZ German Research Centre for Geosciences and gempa GmbH."* (B)

We fit all our requirements:

Seven single-ended 12 bits embedded ADCs
Values available by reading system files
Running under Debian
90$ for the main board
Synchronized by NTP first, later by a GPS receiver on a third-party cape
Buffering datas within seiscomp server
Saving local datas with seiscomp client slarchive
Sending datas with seedlink protocol handled by seiscomp server
Sending older datas with rsync daemon installed on Debian system

Install and setup Debian on the Beaglebone

As the board has no internal storage, installing the system requires getting a system image file available on http://beagleboard.org/latest-images. Then this image has to be uncompressed and copied bit to bit on the system SD card (2Gb minimum). On Linux systems, the commands are like:

```
root@host:~# wget http://debian.beagleboard.org/images/bone-debian-7.4-2014-03-27-2gb.img.xz
root@host:~# unxz bone-debian-7.4-2014-03-27-2gb.img.xz
root@host:~# dd if=bone-debian-7.4-2014-03-27-2gb.img of=/dev/#name of SD card
```

Once the SD card is ready, the BeagleBone will boot on its new Debian system. It is accessible through ssh on the network as it gets an IP through DHCP on its main network interface, or through the Usb port as it creates a virtual interface with an address of 192.168.7.2. See http://www.beagleboard.org for more details.

The first action to do on the new system is creating a new user named sysop, as it is strictly not recommended to use seiscomp under root.

```
root@beaglebone:~# adduser sysop
```

Activating the ADCs and read the values with bash

In order to access the ADCS as standard system files, the user needs to activate a sub-system of the kernel named "IIO" as "Industrial I/O", dedicated to handling ADCs. It can be done by using the "cape-manager" of the kernel. It works very simply by concatenate the name of cape to a dedicated file.

```
root@beaglebone:~# echo cape-bone-iio > /sys/devices/bone_capemgr.8/slots
```

A new directory "helper.11" appears in /sys/devices/ocp.2/. It contains files names AIN[0-9] pointing on ADCs input. Their content is directly readable:

```
root@beaglebone:~# cat /sys/devices/ocp.2/helper.11/AIN0
1127
```

The ADCs are ready to be read by any program able to get datas in a file. It returns a value in millivolts, and its maximum value is 1800mV.

Compiling seiscomp on the Beaglebone

The package is available on http://www.seiscomp3.org/wiki/download. Once copied and uncompressed, its compilation (under user sysop) is quite easy as it is documented and there is an active community working on it. However, it takes a few hours as compilation needs a lot of resources and processor is only 750MHz.

```
sysop@beaglebone:~$ scp user@host:seiscomp3-seattle-2013.340.01-src.tar.gz
sysop@beaglebone:~$ tar zxvf seiscomp3-seattle-2013.340.01-src.tar
sysop@beaglebone:~$ cd seiscomp3
sysop@beaglebone:~/seiscomp3$ make -f Makefile.cvs
```

The opened menu allows check and link all missing libraries. As seiscomp handles a lot of functionality, you may have several software dependancies to satisfy before compiling. To get the missing software, use apt-get, a software utility used by Debian.

```
sysop@beaglebone:~/seiscomp3$ cd build
sysop@beaglebone:~/seiscomp3/build$ make && make install
```

The whole compilation can really take a while. As a compilation takes also a lot of resources, you may need more memory, by adding virtual memory. This is done by creating a file, and activate it as swap:

```
root@beaglebone:~# dd if=/dev/zero bs=$((1024*1024)) count=256 of=/swapfile0
root@beaglebone:~# mkswap /swapfile0 && swapon /swapfile0
```

After compilation, the system is now ready to run seiscomp. Running seiscomp on such a device is absolutely not a problem, as long as we do not use the graphical interface.

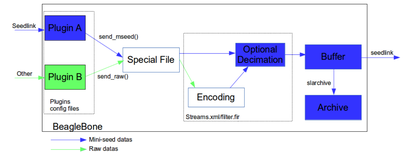
About quickly deploying several stations

As we mentioned before, the image file is copied bit to bit to the SD card with a program such like dd. dd allows copying image file to an other image file too. Each image can be modified on the host computer by mounting it on the local file system. This can be done with the loopdevices under Linux. Once mounted, the files of the future system can be edited, for setting up Ip adddress, station name, and so on. It is then possible to serialize the setup of several Beaglebone from one computer.

```
root@host:~# file bone-debian-7.4-2014-03-27-2gb.img # get the startsector of 2nd partition
root@host:~# losetup -f -o (startsector*512) bone-debian-7.4-2014-03-27-2gb.img
root@host:~# mount /dev/loop0 /mnt/bone
```

How seiscomp does work?

Seiscomp is a complete package with different programs for acquisition, saving, sending datas, but also for processing them. We focus here on the acquisition process. The whole process is made by differents programs with a unique role. Acquisition itself is made by different plugins getting all datas (with seedlink or not) from the dataloggers and make it available on a linux special file (descriptor #63). A plugin can get datas already in mini-seed or in raw format. If datas are already mini-seed encoded, they are directly transfered from the special file to the server buffer, through an optional decimation stage. If they are in raw format, they need to be encoded before being available in the buffer. As all datas transit through this special file, the seiscomp server does not care which system the datas come from while its station name, channel name and location code are advised. Otherwise the datas are ignored. Once the datas are in the buffer, it becomes available for every client like other seiscomp systems or a local client for archiving datas.



Each step has its own configuration:

The plugins are able to run in standalone, each one is configured by parameters within a file. It calls either send_mseed() or send_raw() functions.Those are defined in a C header file provided within the package, allowing home-made plugin development.
The encoding are managed by the files streams.xml and filters.fir. The first one defines the mini-seed parameters for raw-datas and decimation filtering. The second one defines the fir filters used for decimation.
The server responds all request of the clients: datas for other seiscomp systems through seedlink, slarchive requests datas for local archiving, and slinktool is another powerful client for checking, selecting and saving datas.

Development of a home-made acquisition plugin

That plugin has to read datas within the system filesystems as explained before, and use the function send_raw(). We chose not to encode in mini-seed as seiscomp handles an encoder. Here is the main part of the code:

```
void acqProcess(char *station, char* channel, int rate, char *f_source){
     p_file=fopen(f_source, "r");                                   //open file /sys/devices/ocp.2/helper.11/AIN0
     while(1){                                                      //begin loop
          gettimeofday(current_t, tz);                              //get current system time
          for (inc=0; inc<n_sample; inc++){                         //fill little buffer of n_sample
               fscanf(p_file, "%d", buffer+inc) ;                   //reads file
               fseek(p_file, 0L, SEEK_SET);                         //rewind file pointer
               usleep( (useconds_t) (1000000./rate) );              //wait for newt sample
          }
          send_raw_depoch(station, channel, current_t, 0, 100, buffer, n_sample);       //send n_samples to special file
     }
     return;
}
```

Once compiled, the program can run in standalone. It is launched by seiscomp itself, but you can test it to check if datas are acquired:

```
sysop@beaglebone:~/bb_plugin$ gcc -Wall -Wextra -c plugin.c
sysop@beaglebone:~/bb_plugin$ gcc -Wall -Wextra -lm plugin.o main.c -o /home/sysop/seiscomp3/share/plugins/seedlink/bb_plugin
sysop@beaglebone:~/bb_plugin$ cd ~/seiscomp3/share/plugin/seedlink
sysop@beaglebone:~/seiscomp3/share/plugin/seedlink$ ./bb_plugin -f config_file 63>data.dat     #
```

The outputs of file descriptor 63 (descriptor number of linux special file used by seiscomp) is redirected to data.dat. If this file grows up, then the plugin is working correctly

Setting up the plugin within seiscomp

The general configuration of seiscomp is not discussed here, because there are a lot of documentation of getting started with seiscomp. We discuss here on how integrate our home-made plugin in the toolkit. To have seiscomp running, an inventory file has to be imported to define the station and network name. Then modules spread, scmaster, seedlink and slarchive must be enabled and configured to bind the station. Using a home-made plugin requires some special configuration step:
Specify the exact command to launch the plugin at seiscomp startup by editing seedlink.ini
Activate the encoding and decimation by precise the path of files stream.xml and filters.fir
Define a proc name within streams.xml
Here is a brief summary on how we configured it:

seedlink.ini

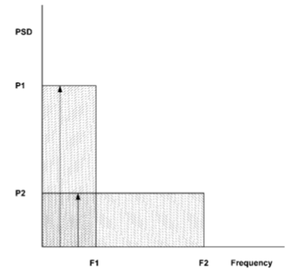
```
     # Activating encoding and filtering
     filters = "/home/sysop/seiscomp3/var/lib/seedlink/filters.fir"
     streams = "/home/sysop/seiscomp3/var/lib/seedlink/streams.xml"
     # Calling newly compiled plugin, it supposes the program will read all needed informations in bb0.ini: source file name, station name,
input channel name, rate
     plugin bb0 cmd = "/home/sysop/seiscomp3/share/plugins/seedlink/bb_plugin -f /home/sysop/seiscomp3/var/lib/seedlink/bb0.ini"
     # Defines what process written in streams.xml the datas will go through
     station GPIL  description = "Grand Pilier"
               name = "GPIL"
               network = "XX"
               proc = "bone"
```

streams.xml

```
     <streams>
          <proc name="bone">                                        #proc name, must fit with proc parameter in seedlink.ini
               <tree>
                    <input name="0" channel="I" location="00" rate="1000"/>     # name and rate must fit with parameters defined in bb0.ini
                    <node stream="CK">
                         <node filter="F260" stream="EK">              #filter must be defines in filters.fir Here: decimation factor 10
                              <node filter="F260" stream="BK">
                                   <node filter="F260" stream="LK"/>
                              </node>
                         </node>
                    </node>
               </tree>
          </proc>
     </streams>
```

That configuration makes 4 different mini-seed streams: GKI at 1kHz, EKI at 100Hz, BKI at 10Hz and LKI at 1Hz. Everything is ready for the first run of seiscomp, by typing "seiscomp start". If a problem is encountered, seiscomp write a log file called seedlink.log

About signal processing

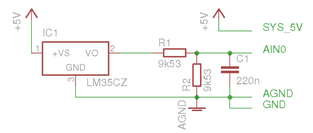


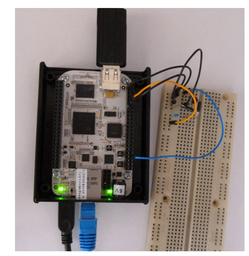
The first (and probably most important) rule of acquisition is to respect the theorem of Nyquist-Shannon, by applying an analogic anti-aliasing filter just before the ADC. Its corner frequency must be below the sampling_rate/2, but has to be designed to not filter our study frequencies.

An other rule is, for a less-noise signal, using the highest possible sampling rate and decimating with Fir filters decreases the influence of electronic noise of ADC. R. Sleeman et al (C) explain this phenomena:
"The next figure comes from One-sided power spectral density (PSD) levels (P1 and P2) for an ideal quantization process using two different sampling rates with corresponding Nyquist frequencies F1 and F2. The area of the rectangle bounded by frequency F1 and PSD level P1 equals the area bounded by F2 and P2. Both areas are equal to the quantization noise power."

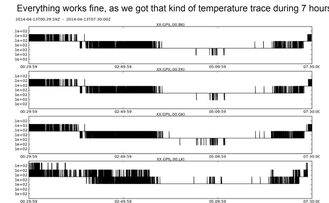
Our first low-cost datalogger

Our first Beaglebone station was designed to get temperature from a LM35 sensor, which outputs a voltage proportional to the temperature: 100mV/°C. The input dynamic of the Beaglebone ADCs is between 0 and 1800mV. This means we can't get temperature higher than 18°C, and wanted to go warmer. We therefore designed the following (and very simple single order) analog adapter. For getting a temperature stream at 100Hz, we will digitize at 1kHz (because of NTP precision is about 1ms), and then decimating to 100Hz (see streams.xml file before). We apply a first order analog filter whose cutoff frequency is defined by R1, R2 and C1 at 179Hz. The gain is defined by R1 and R2 at 0.5. We then have a dynamic up to 0-36°C.



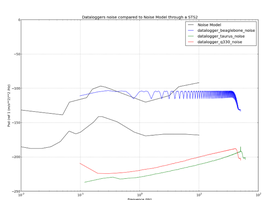


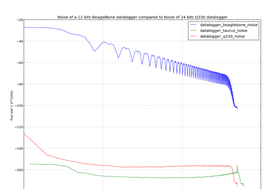
Everything works fine, as we got that kind of temperature trace during 7 hours, with a theorical precision of 0.02°C



About the dynamic range of the Beaglebone

Just for fun, we tried to evaluate if those embedded ADCs could digitize seismological signal. We calculated its eletronic noise by using a method described by R. Sleeman et al. (C) and compared the results to other dataloggers.





As those embedded ADCs are only 12bits and lower range dynamic than the commercial seismological dataloggers, it makes them unusable for study of seismological noise. May be in a few years we would buy a credit-card size computer, with high-range, 24 bits, differential inputs, low-noise embedded ADCs.

References

(A) BeagleBoard.org Foundation. BeagleBone, [On line]. http://beagleboard.org/Products/BeagleBone (page read on the 21st of april 2014)

(B) Helmholtz-Zentrum Potsdam Deutsches GeoForschungsZentrum - GFZ, SeisComP3, [On line]. http://www.seiscomp3.org (page read on the 21st of april 2014)

(C) "Three-Channel Correlation Analysis: A New Technique to Measure Instrumental Noise of Digitizers and Seismic Sensors" by Reinoud Sleeman et al., Bulletin of the Seismological Society of America, Vol. 96, No. 1, pp. 258–271, February 2006, doi: 10.1785/0120050032