Beagleboard Programming

Seems to be three main languages that are used on the Beagleboard: C, Python and Javascript

* C works pretty much as expected since it’s just a Linux machine
* Same with Python, although there are a couple of custom libraries you can download (such as one by Adafruit) that make GPIO access easier
* Javascript can be used through something called Bonescript – this is a custom Javascript implementation that basically imitates Arduino function calls, running either through a browser/webserver combo on the Beagleboard or from a local PC
* C is probably the safest option since it’s familiar and we’re using it for the server as well
* Python on the Beagleboard also seems quite good and is potentially simpler than C
* Bonescript may also be worth looking into more, since it’s already focused around web use (and is similar in syntax to Arduino stuff) – however, there could be issues with timing, complexity, etc. since I’m not sure how fully developed it is or how easy it would be to integrate into other parts of the software.
* One issue with C programming on the Beagleboard is that it seems to be the least used of the three options above; it’s much harder to find examples and actual documentation with C code. I don’t think we’ll run into any problems using it though.
* For C code use the standard compiler/linker that comes with Beagleboard UNIX distribution

Interacting with the GPIO pins and ADCs on the Beagleboard is done through *sysfs* in its Linux OS:

* There are eight AIN files, one for each of the eight ADC interfaces. Using each name, each ADC signal can be opened/read/etc with standard C functions.
* AIN0: /sys/devices/platform/tsc/ain1
* AIN1: /sys/devices/platform/tsc/ain2
* …all the way to AIN7: /sys/devices/platform/tsc/ain8
* These directory paths can vary with the revision number of the Beagleboard, so we’ll have to double-check the manual for this. I’m also not sure which pins link up with which ADCs yet.
* Sample use: int n = open("/sys/devices/platform/tsc/ain2", O\_RDONLY);
* Reading digital values from the shell: #cat /sys/devices/platform/tsc/ain2
* Interacting with other board elements is also done through *sysfs*; i.e. when interacting with an LED: "/sys/class/leds/beaglebone:green:usr0/brightness"
* Can also address GPIO pins directly this way: "/sys/class/gpio/gpio%d/attribute"
* Things to note: digital values from the ADCs must be converted to millivolts, then to pressure (or temperature or whatever the sensor is measuring)

We can do a lot of simple testing and debugging through shell commands, for example:

* cat /sys/devices/bone\_capemgr.\*/slots to list current peripherals
* cat /sys/bus/iio/iio:deviceX/in\_voltageY\_raw to check voltage value
* The standard Beagleboard has an am355x IO driver with a test application called generic\_buffer.c that reads the sensors.
* For continuous sensor sampling: can create a trigger via the shell, connect this to the adc drivers, select a channel to use and sampling rate. Straightforward commands.
* A lot of this is dependent on Beagleboard documentation and actually testing this on the hardware (and we don’t know when it will arrive).
* <http://circuitco.com/support/index.php?title=BeagleBoneBlack> has a bunch of useful information and documentation links
* Also see derekmolloy.ie/beaglebone/ for tutes (including some networking, OpenCV and video streaming stuff). Seems useful.

A couple of simple examples and the Beagleboard test drivers are included in the Git folder.