Reactive Motor Controller Manual

Caitrin Eaton

December 5, 2016

Contents

Part I
User's Guide

1 What the Reactive Motor Controller does

The Reactive Motor Controller is designed to interface with the Aurora Scientific suite of voltage-controlled motors. It accepts an input voltage that is assumed to represent the current state of the motor (usually `FORCE OUT`), and generates an output voltage that can be used to command the next state of the motor (usually `LENGTH IN`). The overall effect is that of a voltage spring: $V_{out} = k_V * V_{in}$, where V_{in} is the voltage that is read in and V_{out} is the generated output voltage. When V_{in} is tied to `FORCE OUT`, and V_{out} to `LENGTH IN`, the position of the Aurora Scientific motor will be adjusted as a function of force such that it acts as a mechanical spring. The spring constant is defined in software and can be adjusted between trials.

2 Requirements for use

2.1 Voltage source

Once programmed, the Reactive Motor Controller requires a voltage source with +15 V, -15 V, +5 V, and GND outputs. The current draw is relatively low (10-20 mA), and a source rated at 0.25 W power should be more than adequate.

2.2 Computer with powered USB

A computer running Linux or Windows is required in order to program the BeagleBone and to send commands to be executed on the BeagleBone. This communication happens over powered USB. If the USB port is not powered, the BeagleBone will still draw enough current to turn on and light up, but will not properly support communication, command execution, or digital I/O. When multiple powered USB ports are available, not all are guaranteed to work. The logic behind this is unclear to your humble author. But, if the BeagleBone seems unresponsive even after drivers have been installed, or attempts to establish an SSH connection repeatedly result in errors, then try plugging the BeagleBone into other powered USB ports on the same machine. So far, our lab has not had any problems that could not be solved by a USB round-robin or driver installation.

2.3 Data acquisition

The provided software does not log data on the BeagleBone, so an external data acquisition (DAQ) system, e.g. one of National Instrument's NI DAQ systems, can be used to monitor `LENGTH IN` and `FORCE OUT` voltage signals. These can then be visualized and recorded on your computer with a data analysis program like Igor, LabView, or MATLAB.

3 Installing BeagleBone drivers on a new computer

If the computer has never been connected to a BeagleBone before, you may have to install some drivers. Instructions are included below. *Note: If you're using Linux, no drivers need to be installed.*

1. Connect the BeagleBone via *powered* USB to a computer with an Internet connection.

2. Open a file browser window and navigate to the BeagleBone folder, which will appear as an external drive, similar to a USB memory stick.

3. Right click `START.htm` and select `Open with Chrome`. (Internet Explorer will not work.)

4. In the page that opens, there should be three clearly marked installation steps on the left side.

5. `Step 1: Plug in BeagleBone via USB` should be highlighted in green with a check mark. If not, the BeagleBone may not be connected to a powered USB.

6. Left click `Step 2: Install drivers` and follow the instructions in the provided table as appropriate for your operating system. These drivers let your computer talk to the BeagleBone. If installation is unsuccessful, the BeagleBone may not be connected to a powered USB.

7. After installing drivers, restart your computer.

8. Re-open `START.htm` in Chrome. `Step 2: Install drivers` should be highlighted in green with a check mark.

9. `Step 3: Browse to web server on board` is unnecessary. There are some interactive scripts there that are helpful if you'd like to learn about general BeagleBone programming, but that's not necessary in order to use the Reactive Motor Controller or edit the spring constant.

You can now program the BeagleBone from this computer by establishing an SSH connection.

4 Establishing an SSH connection

An SSH connection makes it possible to program and control the BeagleBone Black from any computer with the proper drivers (see Section 3), similar to working in a Remote Desktop environment.

4.1 On Linux

To establish an SSH connection from a Linux terminal:

1. Connect the BeagleBone via *powered* USB to a computer with drivers already installed.

2. Open a terminal window.

3. Enter into the terminal: `sudo ssh 192.168.7.2 -l root`

4. You will probably be prompted to enter the computer's password. This is the regular password for your computer. For what I assume are paranoid Linux-hacker security reasons, no characters will appear as you type. Simply enter your password as usual and then hit `ENTER`.

5. If SSH does not succeed, the BeagleBone may not be connected to a powered USB.

6. If SSH succeeded, the terminal is now running on the BeagleBone, rather than your computer. Any commands you enter will be executed on the BeagleBone. For starters, list the contents of the current folder with the command `ls`.

4.2 On Windows

To establish an SSH connection from PuTTY on a Windows machine:

1. Make sure PuTTY is installed. If not, download it for free from www.putty.org.
2. Open PuTTY.
3. In the IP field, enter: `192.168.7.2`
4. Make sure the `SSH` option is selected. (I believe the default is Telnet, which will not work here.)
5. A terminal window will open, connected to the BeagleBone. When it gives you the login prompt, type `root` and hit `ENTER`.
6. If the window does not appear or gives you an error message instead of a login prompt, the BeagleBone may not be connected to a powered USB.
7. If SSH succeeded and you have logged in as `root`, the terminal is now running on the BeagleBone, rather than your computer. Any commands you enter will be executed on the BeagleBone. For starters, list the contents of the current folder with the command `ls`.

4.3 CAUTION: Safely disconnecting the BeagleBone

When you are done, it is important to enter the command `poweroff`, so that the BeagleBone shuts itself down before it is unplugged from the powered USB port. Once all the lights have gone off, it is safe to close the terminal window and unplug the BeagleBone.

5 Programming the BeagleBone

The BeagleBone can be programmed from an external computer through a terminal window in which an SSH connection has been established. This must be the same terminal window in which you have logged in to the BeagleBone. Should you accidentally close this terminal window, re-connect by following the steps in Section 4. Remember that the terminal is running on the BeagleBone, which has a Debian Linux operating system. If you're feeling stuck at any point using the terminal, it might be helpful to do an online search for Debian commands. The provided code is written in C++, which is also very well documented with free online educational resources.

The Reactive Motor Controller software is assumed here to have been provided in the file `ReactiveMotor.cpp`.

5.1 CAUTION: The Typocalypse

Be very careful with spelling in all of your interactions with the terminal. Double check all commands for typos before hitting `ENTER`. The terminal will execute *exactly* the command you enter, regardless of what you may mean to do, sometimes with frustrating results. The terminal has no spellcheck, autocorrect, or common sense. For a modern human, this feels a lot like having your smartphone replaced by a sextant and carrier pigeon. Be vigilant. Thar be dragons.

5.2 Configuring BeagleBone I/O pins

The I/O pins on the BeagleBone must be configured before running Reactive Motor Controller code. This is done by loading the included configuration files with the commands

```
echo spi0-loop > $SLOTS
echo ADAFRUIT-SPI0 > $SLOTS
```

where the 0s in `spi0-loop` and ADAFRUIT-SPI0 are zeros, not letter Os. The O in `$SLOTS` is the capital letter O, not a zero. The order of these commands is also very important. Load `spi0-loop` first, ADAFRUIT-SPI0 second.

Pin configuration only needs to be done once after the BeagleBone starts up. Code can then be executed as many times as you like without reconfiguring the pins again until the next time the BeagleBone starts up.

5.3 Backing up code

It is a good idea to copy this code into a new file before editing, so that you always have a backup. In order to copy the original file `ReactiveMotor.cpp` into a new file named `ReactiveMotor_Backup.cpp`, enter:

```
cp ReactiveMotor.cpp ReactiveMotor_Backup.cpp
```

For more information on file handling in Debian, e.g. how to create and copy into different folders, do an online search for *Debian terminal file handling*.

In order to restore your code from the backed-up copy, use the same `cp` command:

```
cp ReactiveMotor_Backup.cpp ReactiveMotor.cpp
```

You may want to give your new code file a distinctive name in the place of `ReactiveMotor.cpp`. Feel free to put any name here, as long as it ends with the extension `.cpp`.

5.4 Editing and compiling code

To edit code, open it in the text editor "nano" using the command

```
nano ReactiveMotor.cpp
```

where you can replace `ReactiveMotor.cpp` with the name of any existing file you would like to edit. Be forewarned that if you misspell the file name, nano will create a new (empty) file with the name that matches your command, and the editor will open with an empty screen. Return to the command line (regular terminal) by exiting nano with the command `CTRL+x` and try again.

With your code open in nano, be aware of nano's custom commands, which listed at the bottom of the terminal window. In this list, the caret symbol ^ is used to represent the control button `CTRL`. The most useful of these commands are:

- Save is `CTRL+o` (written ^o in the nano command list)
- Exit (return to the command line) is `CTRL+x` (written ^x in the nano command list)
- Scrolling and cursor movements are done with arrow buttons, not with the mouse.
- Copy and paste are done with the mouse's right-click menu. Windows shortcuts like `CTRL+c` and `CTRL+v` will not work.

Note that nano will not save your work automatically on exiting, so you must save with CTRL+o before returning to the command line.

After editing your code, you will have to recompile it. Otherwise the BeagleBone will continue to execute the version of your code that was most recently compiled. To compile the newly edited ReactiveMotor.cpp:

```
g++ ReactiveMotor.cpp GPIO.cpp -o ReactiveMotor.o -pthread
```

If your code file has another name, replace ReactiveMotor.cpp and ReactiveMotor.o with the correct file name. The extensions .cpp and .cpp should stay the same.

5.5 Executing code

In order to run your compiled code, enter

```
./ReactiveMotor.o
```

where ReactiveMotor.o can be replaced by the name of your compiled .o file.

6 Guide to the code

To do.

Part II
Fabrication

This section explains how to fabricate a new Reactive Motor Controller.

7 Parts

Parts required to construct and control the custom PCBs are listed here in the format <name: description (quantity)>.

- Adafruit BSS138: 4-channel I2C-safe bi-directional logic level converter (2)
- ADS7813P: Low-power, serial 16-bit sampling analog-to-digital converter (1)
- AD7849CN: Serial input, 14-bit/16-bit digital-to-analog converter (1)
- AD588BQ: Multiple output, high precision, dual-tracking reference (1)
- 450-650 Ω resistor: through hole (1)
- 39 kΩ resistor: 1% tolerance, through hole (1)
- 100 kΩ potentiometer: 10-turn, high precision (2)
- 0.01 μF (10 nF) ceramic capacitor: rated for over 15 V (1)
- 0.1 μF ceramic capacitor: rated for over 15 V (1)
- 1 μF ceramic capacitor: rated for over 15 V (1)
- 1 μF electrolytic capacitor: rated for over 15 V (2)
- 10 μF electrolytic capacitor: rated for over 15 V (1)
- 1N4148: Schottky diode (1)
- 1N5711: Schottky diode (1)
- LED: red (1)
- Push button (1)
- Ribbon cable: 12- or 14-pin, 6", female/socket (1)
- Pin header: 12- or 14-pin, 2-row, female (2)
- Pin header: 12- or 14-pin, 2-row, male (2)
- Screw terminal: 3-pin, 0.137 in (3.5 mm) spacing (4)
- Spacer: 0.5"-1", fits 4-40 screw (8)
- 4-40 screw: 0.25"-0.5" (8)

- Custom PCB: plans attached (1)
- Adafruit BeagleBone Black Proto Cape (1)
- Insulated wire: 22 awg (as needed)

Also required to interface with BeagleBone and Aurora Scientific control box:

- Power source with $+15$ V, -15 V, $+5$ V, and GND terminals
- Aurora Scientific motor control box with 300-series motor
- Independent data acquisition system, e.g. NI DAQ
- Computer with powered USB port, used to SSH onto the BeagleBone
- So many BNC cables (to connect Aurora with the data acquisition system and custom voltage control PCB, and to connect the power source to the custom voltage control PCB)

8 Circuit design

The reactive motor circuit is designed to convert an incoming ± 10 V force signal from the Aurora Scientific motor control black box into digital values with the ADC chip, and to generate an outgoing ± 10 V position control signal with the DAC chip. The PCB design is sketched out in Figure 1

The BeagleBone Black handles ADC readings and DAC commands over an ISP bus. Code running on the BeagleBone Black processes the force reading and computes a desired position command. The custom cape in Figure **??** handles digital communications between the BeagleBone and voltage control PCB (Figure 1).

9 Room for Improvement

- Cleaner power source
- Redesign PCB to reduce electrical noise
- A digital filter with more logic and less duct-tape-and-spit
- Take advantage of BeagleBone's PRU for higher control loop frequency. Aurora's delay will still be a limiting factor; read/write to motor takes approximately 3 ms. Also, collaborators may have a tougher time reading and editing in the PRU's assembly language.
- Switch to a non-Aurora system that doesn't require this Rube Goldberg device.

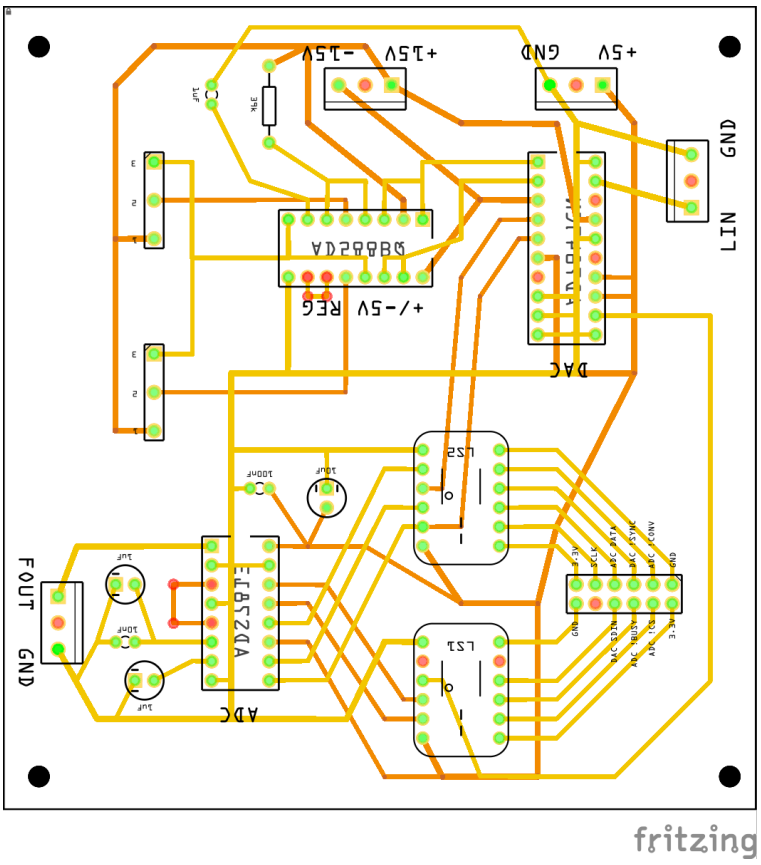

Figure 1: Reactive motor custom PCB for ADC/DAC interface with Aurora Scientific's motor control system.

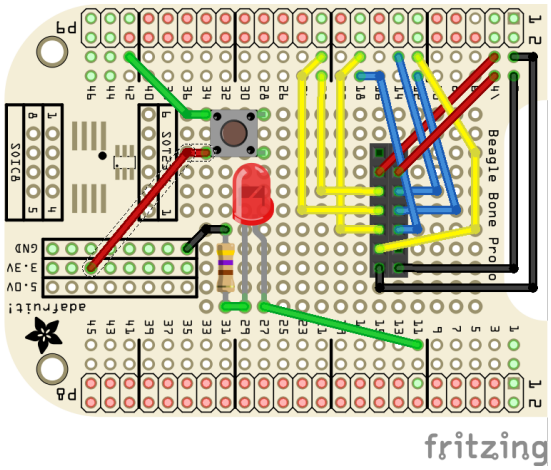

Figure 2: Custom BeagleBone cape for BeagleBone-ADC/DAC interface.

Part III
Cheat Sheet

SSH connection

- BeagleBone's IP address: `192.168.7.2`
- login name: `root`

BeagleBone terminal commands

- list current folder's contents: `ls`
- copy code: `cp <original_file_name.cpp> <new_copy_name.cpp>`
- edit code: `nano <file_name.cpp>`
  - scroll: arrow keys
  - select a section of code: hold `SHIFT` while scrolling with arrow keys
  - copy: mouse right-click menu
  - paste: mouse right-click menu
  - save: `CTRL+o`
  - exit: `CTRL+x`
- compile code: `g++ <file_name.cpp> GPIO.cpp -o <file_name.o> -pthread`
- execute code: `./<file_name.o>`
- safely disconnect BeagleBone: `poweroff`
- close terminal window: `exit`

Plug your own file names in for terms marked with triangular brackets <>. Do not type the brackets.