**Guide to Setup and Use the WAVES Wireless Communication Subsystem**

**Installing and Configuring the Adafruit Feather M0 with LoRa**

Both the AUV and shore controller radios use the same Arduino sketch, and both will need u.FL connectors for their antennas soldered to them in the appropriate footprint. The radio to be used in the AUV will additionally need two 2.2k resistors soldered directly: one from pin USB to A2, and one from A2 to GND. There will also be wires from GND, Tx1 and Rx0 to the Beaglebone black that need soldering. Please consult the full system wiring diagram to ensure this is correctly executed, as inverted UART connections will lead to nonfunctional communications.

(This is essentially a summary of the instructions available from Adafruit here: <https://learn.adafruit.com/adafruit-feather-m0-radio-with-lora-radio-module/setup>)

In order to program the Adafruit Feather M0 using the Arduino IDE, one first needs the Arduino IDE. Once the most updated version is installed, go to File->Preferences and add to the “Additional Boards Manager URL” this address:

https://adafruit.github.io/arduino-board-index/package\_adafruit\_index.json

If other URLs are already listed, simply add a comma and then the above (as with PATH additions).   
  
Next, go to Tools->Boards->Board Manager and select from the “Source” dropdown menu the “Contributed” section and install the “Adafruit SAMD Boards” and “Adafruit AVR Boards” and “Arduino Leonardo and Micro MIDI-USB” libraries. Make sure to FIRST install any Arduino Boards library which has inclusions for the Feather (Arduino SAMD and Arduino AVR). Restart the IDE, then plug in the Feather M0 and verify that the Tools->Ports Menu shows “COMxx: Adafruit Feather M0” for xx whichever USB port you are plugged into. Select the correct board from Tools->Boards. Now, you can install a sketch onto the Feather! There is a sample Tx sketch and a sample Rx sketch on the Adafruit guide above which can be monitored from the Arduino serial monitor to verify that the two boards are pinging each other. DO NOT TRY TO USE THE RADIOS WITHOUT ANTENNAS. Either use a 13cm wire or your intended antenna.

The first lines of code below the library imports on the AUV\_Comms Arduino sketch allow the toggling of modes. The appropriate version of the sketch is obtained by compiling and uploading after ensuring the proper toggles are either uncommented or left commented out. A guide to these possible settings is as follows:

|  |  |  |
| --- | --- | --- |
| **#define SHORE\_CLIENT** | **#define DEBUG** | **USE CASE** |
| commented | commented | Official AUV installation with BBB. All arduino interaction will be over UART with no extra information or added UX dialogue. Validate both radios operational with DEBUG modes and USB terminal before switching to this setting. |
| uncommented | commented | Not a useful state. Avoid. |
| commented | uncommented | Troubleshooting AUV radio via USB with similar terminal interface as used for Shore controller radio. |
| uncommented | uncommented | Official shore controller radio setting. Use with USB serial interface. |

**Available Radio Methods:**

The serial input is fairly reliable and allows backspace, so it should be fairly easy to interact with. Whatever you type will, as long as the transmission executes properly and is received and acknowledged, be relayed verbatim to the AUV Beaglebone Black from the perspective of root kernel access. This link is not suitable for interactive programs or lengthy output, and not all command line operations will behave predictably, but most may be performed by preceding your command with “sudo sh -c” to spawn a child shell from which to execute your desired operation.

The following special commands do not result in AUV interaction but govern various available radio options:

“-changemode bw500cr45 sf128” gives the syntax form for a coordinated and handshaked settings change on both radios. All settings options must be specified, with the options listed and explaiend in the next section.

“-changeme bw500cr45 sf128” allows a local-only settings change. In the event that the shore radio loses power or is reset, the same settings can be implemented to regain the link, but there is user responsibility to remember the settings they were using. Once a preferred setting is found for a specific mission’s operating territory and link conditions, it is recommended to recompile and reupload the sketches with those settings as default.

“-verb” gives verbose output to monitor link conditions, appends all received messages with total packet time from sending to receiving acknowledgement and the SNR and RSSI detected. If packet times far exceed those expected for the settings in use, it may be assumed that some retries were necessary.

“-memmon” can be used in sketch modification to check memory use over time when new features are added. If the new features cause memory leak, this may be determined by invoking this method and observing memory use changes over the course of an interaction section.

“PING” is a simple ping to test that 2-way communication is working without any AUV interactions. The response comes from the Arduino, not the BBB, and it can be convenient to check the link by turning -verb on before a ping, then turning -verb back off.

**Choosing Radio Settings**

The “Different LoRa Settings” Spreadsheet gives a rough guide to the datarate differences between the different available settings. Here are the meanings of those settings.

We have provided access to three of the available chirp bandwidths: 125kHz, 250kHz, and 500kHz. The smallest bandwidth is thought to offer roughly 6dB improvement in link budget (theoretically a doubling of range) but with proportionally longer packet time, about 4x greater on-air time in tests.

The coding rate sets the Hamming Code depth, from 4/5 to 4/8. The 4/5 setting only enables a single parity bit, while the 4/8 setting can detect multiple bit error locations. This setting is exclusively related to reliability and does not affect link budget, but on-air time increases slightly. This option is somewhat redundant due to the reliable datagram’s CRC and so it will not dramatically improve performance nor will it significantly hurt on-air time. It is recommended to leave at 4/5 except in specific instances.

The spreading factor specifies chips/symbol, where symbol rate is set by adjusting the number of phase changes (chips) used for each symbol. A high spreading factor results in greater on-air time but improved detection and decoding. Each increase in spreading factor doubles the on-air time but can potentially provide around 2dB improvement in link budget. The available spreading factor options are 128,256,512,1024,2048,and 4096. It is recommended to limit spreading factor to 2048 at most, as 4096 can lead to packet times up to 8s and it is likely that they will time out. The timeout setting can be adjusted in the sketch, but failed packets at faster datarate are also subject to it.

Current Default settings in the sketch:

BW: 500

CR: 4/5

SF: 256

# RETRIES: 3

TIMEOUT: 4 seconds

Each retry attempt is given the full timeout period if no acknowledge is received. This means packet failure can be detected for certain within 16 seconds of initial transmission.

Recommended default settings for poor link environments:

BW: 500

CR: 4/5

SF: 2048-4096

Estimated Packet time: 0.7-1.2s

Since the AUV is floating, it is as likely that the RF link is broken mid-packet as not and so the lowest possible spreading factor will yield best performance due to the low on-air time. However, physical link budget is 6dB worse at the higher bandwidth, which makes it something of a runoff between the following set of settings as far as possible range is concerned.

BW: 125

CR: 4/5

SF: 512 - 2048

Estimated packet time: 0.8-2.7s

**Installing the Communications Software on the Beaglebone Black**

1. (If starting with new Beaglebone Black) :

Flash a Debian 9.5 or earlier image onto BBB:

Create uuuv user with “sudo adduser uuuv” then make a superuser with “sudo usermod -aG sudo uuuv”. Check that it worked by “su -uuuv” then run “sudo whoami” and make sure root shows up. Unzip riptide installer package to some directory, then run ras-uuuv-dashboard-psreinstall.sh and finally install.sh, selecting 1MP, Mk. 2, Sparton IMU as the install options.

1. Log in, then:

sudo nano /etc/network/interfaces

Append to end of doc:

iface usb0 inet static

address 192.168.7.2

netmask 255.255.255.252

network 192.168.7.0

gateway 192.168.7.1

dns-nameservers 8.8.8.8

post-up route add default gw 192.168.7.1

In terminal again:

sudo reboot

Share your internet connection from “networks and sharing” in your OS, then ssh into the BBB which should now be internet-enabled to get more software up on that thing.

Once rebooted:

sudo ntpdate pool.ntp.org

sudo apt-get update

sudo apt-get install minicom build-essential python-dev python-pip -y

sudo apt-get install watchdog

sudo apt-get install python-systemd

sudo pip install Adafruit\_BBIO

sudo pip install pyserial

NOTE: depending on the default python, it may be necessary to change the systemd unit file to direct the process to be run from /usr/bin/python3 or /usr/bin/python2.7 test the script by running it on demand instead of as a service before enabling it in systemctl.

In /home/debian, place the pyTerminal.py file.

Scp ./pyTerminal.py debian@192.168.7.2:/home/debian

In /etc/systemd/system/ place the pyTerminal\_service.service file. You will need to either scp as root or temporarily place it as a user then move it with sudo privilege.

In /etc/systemd/system.conf uncomment the lines relating to watchdog timers and change the RuntimeWatchdog to 15 seconds.

Then from the command line, verify that systemctl can find the unit file with:

$ systemctl list-unit-files | grep pyTerminal\_service

And see the output:

pyTerminal\_service.service disabled

Finally: release it to control of systemctl

sudo systemctl enable pyTerminal\_service

Sudo systemctl daemon-reload

Sudo systemctl start pyTerminal\_service

Check that it started and reported running correctly

Sudo systemctl status pyTerminal\_service

BEFORE ENABLING AS SERVICE:

Run as simple python script with comm system connected and verify that some commands work correctly. Try an ls, etc. It is possible that some debugging and troubleshooting will be required to implement this if your kernel and/or python installations deviate from the dev system. Sorry about that.

**IMPORTANT: OTHER WARNINGS**

Some commands may not work correctly since the service runs from the kernel (not a shell). If shell code is essential, must try “sudo sh -c” before the command to spawn a child process shell to run it from.

Because Lora is not full-duplex, any process output initiated over LoRa will try to continue to send until complete. It is the responsibility of the user to prevent the system from trying to send long files or entering processes requiring interactivity. Due to the way commands are directed as a subprocess, it is likely that the usual means of suppressing “full” output like grep and and tail will not work. Full exploration of the capabilities are recommended before attempting anything but simple file IO.

**The Most Important Outstanding Unfinished Tasks:**

As it is, the correct commands to toggle the sonar on with the correct settings need to be found. The script itself is locally called on the payload computer by running “/home/wracksweeper/waves.sh” but addressing the payload computer correctly at the AUV’s ethernet port and sending this command has not been successfully accomplished due to time limitations.

Additionally, the correct command line instructions are needed by which a GPS current nav-data structure can be stored as a file, then parsed and built into a convenient single line for transmission and/or storage on the Arduino as a “last known coordinates” setting. It is believed that the uXMS feature of MOOS must be exploited for this, or perhaps a custom AppCast set up. Once this is done, the Arduino sketch and the pyTerminal service must both be modified to allow for regular transfer of last-known GPS data to the Arduino so it’s available on demand.

**Improvements that need making:**

Run the terminal service from uuuv instead of root. This will require access to the uuuv or an identical virtual machine to verify correct operation.

There is a scheme by which “esc” can be sent which will terminate output, but the messages never arrive send both radios are sending and neither is listening. Instead, a timeout may be implemented that prevents outputs with longer than 30s of data unless a keyword is included at the head of the command. The “esc” function may be possible by use of the waitCAD() or isChannelActive() class members, but early attempts to implement these methods were not successful.

The reliable datagrams can also be made secure and encrypted by invoking the RHEncryptedDriver class. This should be considered high priority due to the nature of the RF link, but as it represents nontrivial change to the functionality of the current software and requires additional crypto libraries it must also be assessed for its effects on system performance and it will only be a recommendation for future work. <https://www.airspayce.com/mikem/arduino/RadioHead/classRHEncryptedDriver.html#aee2e3ca52262f1b22297d4722fb4c66a>

High priority for future work on this system would entail the creation of a terminal emulator GUI which offers a command line but also buttons and pulldown menus to access the “known safe” common features and the option to export received data to a local MOOS instance for mapping either once or continuously. Logging and interpretation scripts would also be extremely useful. A means of generating MOOS mission update requests that are callable with a single command line entry would extend the level of overall control significantly, as would a method of algorithmically reviewing sonar data and isolating potential points of interest to be able to report to shore if anything worth looking at was found. Machine vision techniques are likely required for this, but the payload BBB offers the available compute power without additional hardware overrhead.

Other future-facing improvements: Additionally, a backup GPS device may be installed which is slave only to the MCU regulating wireless communications, offering redundant asset tracking which can be correlated against beacon data if system failure occurs and MOOS offers the API to facilitate joystick-based piloting, which could be easily added to the existing hardware and improve the launch and recovery process significantly. The MCU governing communications could also be exploited to incorporate leak detection sensing and engage some method of emergency protocol for the AUV.