Tyler Eaves

UCSD CubeSat

This document covers the SPI between a Raspberry Pi and a SX1278 LoRa transceiver module. The relationship between pin purposes, numbers, labels, and wire colors are listed below for convenience. The SX1278 radio is interfaced via the “inAIR4” breakout board with solder masked pin names.

|  |  |  |  |
| --- | --- | --- | --- |
| *Line Purpose* | *RPI Pin #* | *inAIR4 Pin Name* | *Wire Color* |
| 3.3V Power | 17 | 3V3 | red |
| Ground | 25 | 0V | **black** |
| SPI MOSI | 19 | SI | yellow |
| SPI MISO | 21 | SO | orange |
| SPI Clock Signal | 23 | CK | blue |
| Chip Select | 24 | CS | green |
| Reset | 16 | RT | **grey** |

To access the Raspberry Pi’s supported SPI library, *bcm2835-1.56*, open the PI’s terminal window after boot and enter the following command:

$ *cd /usr/src/bcm2835-1.56*

In this directory you’ll find everything you need, from library source and header files to examples. For the SPI test code follow this path:

$ *cd examples/spi*

First you’ll need to compile the source code via:

$ *cc spi.c -o spi -lbcm2835*

Run the following command to execute the test program and verify the SPI configuration, assuming you’ve connected RPI pins #19 and #21 using a jumper wire:

$ *sudo ./spi*

Located back in /bcm2835-1.56/src are the header and source files for the library. You’ll want to examine the .h file for detailed instructions on how to use each function.

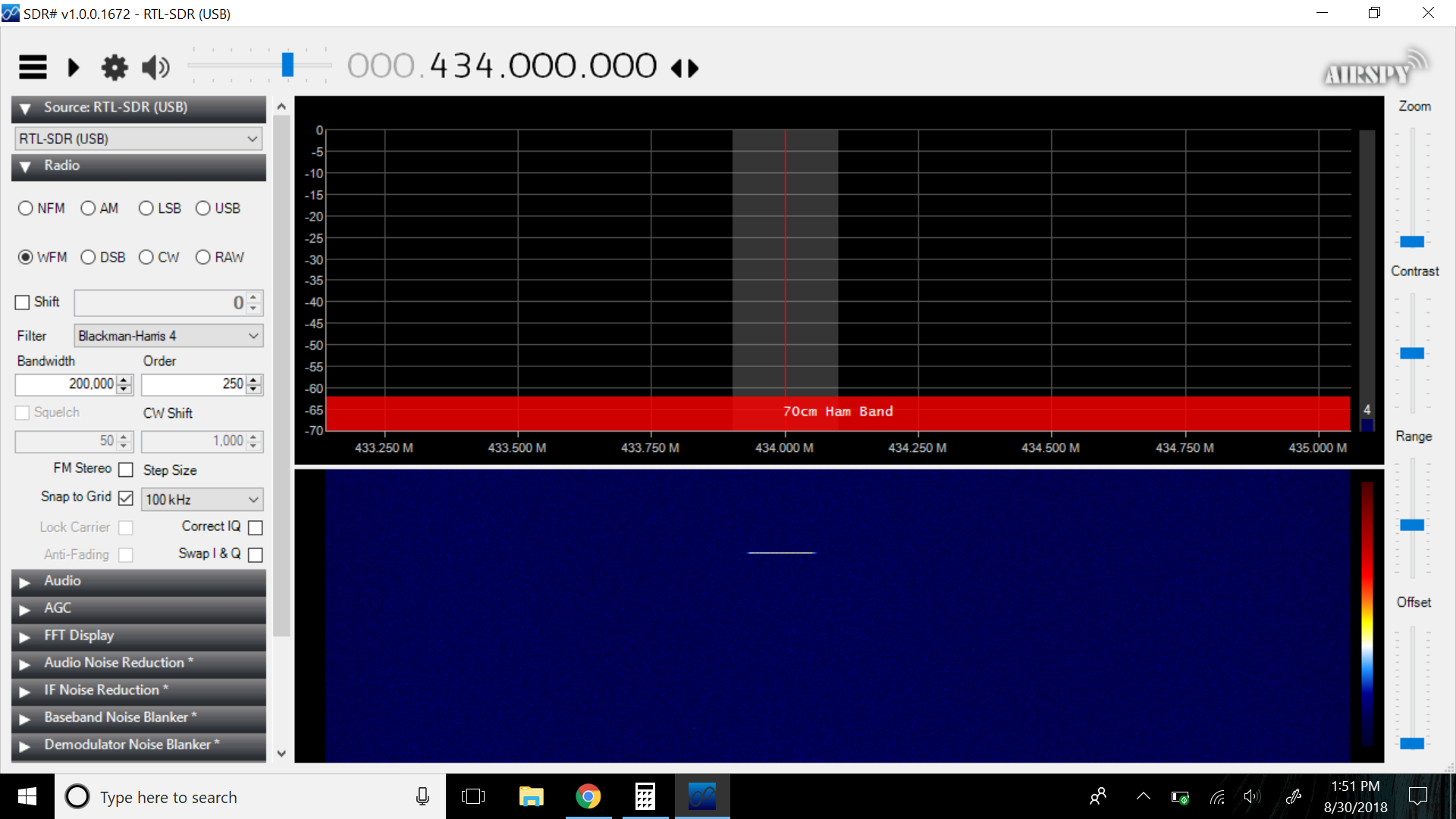
I’ve since written some test programs to configure the SX1278 chips as transmitters and receivers. Assuming the correct file exists on the raspberry pi you’re using (they can all be downloaded from this drive folder) you can execute them with either:

$sudo ./loraTX

OR

$sudo ./loraRX

You will be able to observe packets printed to the terminal window on the receive side. Another cool way to observe LoRa traffic is by running the SDR# program in conjunction with an SDR dongle. Tune to exactly 434.000 MHz and you’ll see a periodic flare in the spectrum. You can also hear the transmission “chirp” if you turn up the audio. You’ll want to set your demodulator to wideband FM with a bandwidth of about 200 kHz. Below is the LoRa image you’ll see on SDR#’s spectrogram:



Yes, that little yellow line is all you’ll see. It’s working correctly. This image was taken with SDR# paused so you don’t see the noise floor or power spectrum, but in real time you’d see short burst peak corresponding to this yellow mark.

For convenience we’d like launching the RPI payload to be as simple as flipping a power switch or plugging in a battery, which means we need to configure it to run on boot. There are several ways to do this, but the simplest is to script it via the RPI’s *rc.local* file. Boot the RPI, open the terminal, and run the following command to open the file with root editing privileges:

$ sudo emacs /etc/rc.local

There should be some notes here about how the script has no default functionality, and there should be an example script that prints an IP address. Regardless, ignore everything and prepare to enter a command somewhere above the line *exit 0*. To configure the LoRa beaconing program to run on boot write the following line:

sudo /usr/src/bcm2835-1.56/loradev/loraTX &

Note that this is just the command you would enter from your base directory to run this program. All we’re doing is scripting it, so almost an exact copy. Just keep in mind that if your program happens to be in a different location, or is named something else this command will obviously read differently. The lone difference is the inclusion of the ampersand symbol at the end of the line. This is necessary because we know this program runs in an infinite loop. The *&* allows the RPI to multitask and continue to boot despite the fact that our program never really “finishes”. To quickly test this script go ahead and run:

$ sudo reboot

Set up a LoRa receiver and RPI running *loraRX* across the room and you should pick up the packets. Additionally you can check your current processes in real time on the RPI running the beacon, and filter out *loraTX* with the command:

$ ps axg | grep loraTX

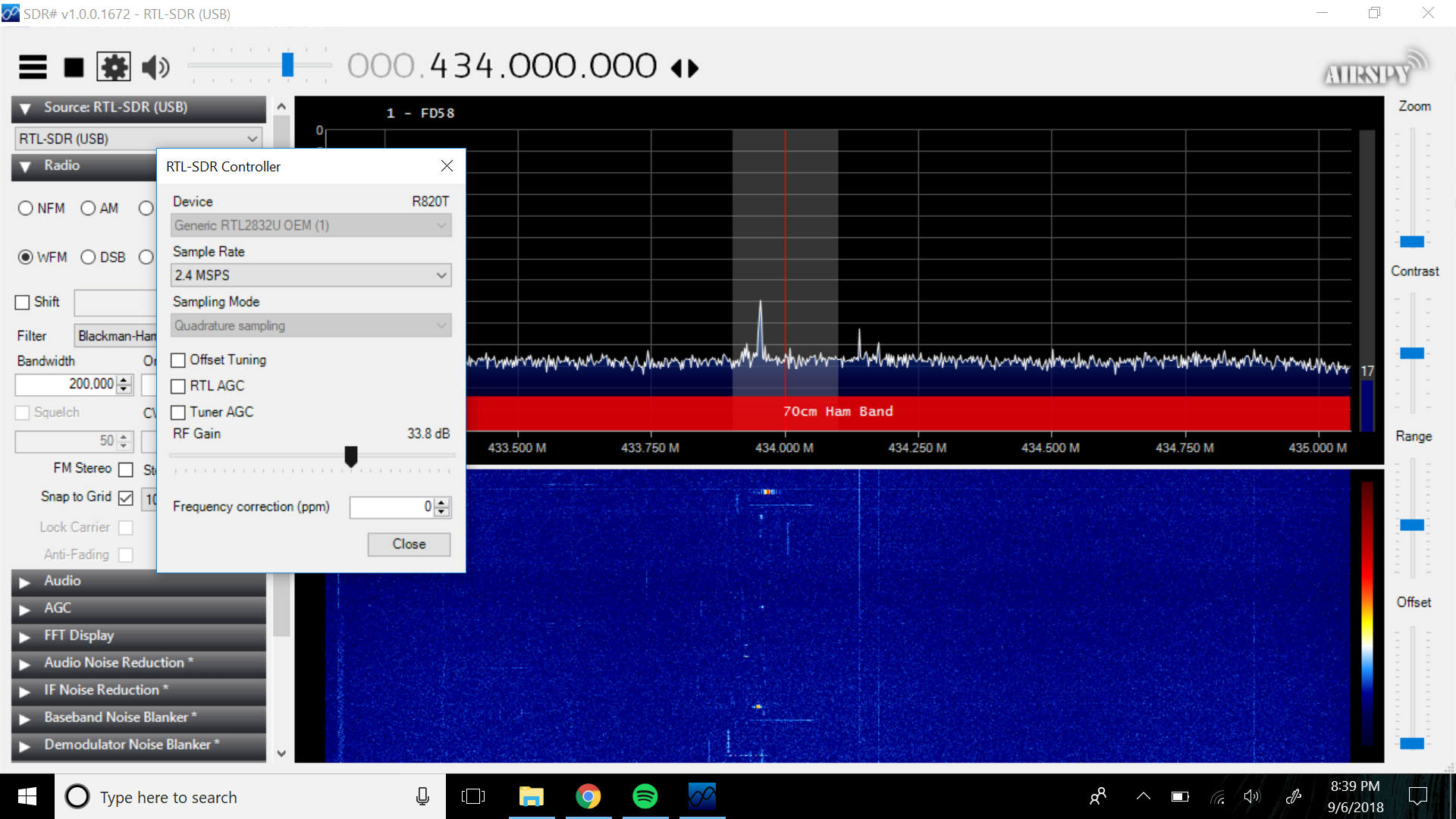
This will return the command we’ve scripted as a process. An optional note, you can redirect program output to a file for logging purposes by appending some modifiers to the script command in *rc.local*. If you instead write the following:

sudo /usr/src/bcm2835-1.56/loradev/loraTX >> /usr/src/bcm2835-1.56/loradev/batlife.txt &

Note that there is no “$” symbol indicating the shell prompt, because this line is scripted! The output stream, stdout, will be appended to the file specified in the path following the “>>” symbol. Using only one arrow will overwrite the file rather than add to it; it’s a matter of preference. In some cases, depending on your PI and other unforeseeable circumstances you may notice the “stdout” stream is not being directed to your file in real time. To fix this modify the above command to the following:

sudo stdbuf -o L -e L /usr/src/bcm2835-1.56/loradev/loraTX >> /usr/src/bcm2835-1.56/loradev/batlife.txt &

The -o and -e flags indicate output and error respectively, and the “L” commands flush the those elements from the standard buffer after every line.

Below is a spectrogram of the LoRa packets I received from my beacon ten miles away! Compare this to the image above. The packets, at five second intervals, are the two very thin horizontal chirps at 434 MHz. Of course these are lower power than the packet received from ten *feet* away, and therefore relatively low power. The transmitter bandwidth setting is 125 kHz which is why the chirp occupies just over half of the 200 kHz wideband FM scan centered at 434 

MHz.

While I’ve been validating the communications side of the operation, Zhaowei has developed software that implements pseudo multi-tasking capabilities on board the RPI. We have temperature sensors and an IMU each accompanied by a python script to survey their behavior. For everything to function properly remember to enable one-wire protocol and I2C on the RPI as these are the means of communication with the temperature sensors and IMU respectively. At boot the startup sequence should first establish the socket, then run the sensor scripts, then attempt to read from the socket with loraTX.c.