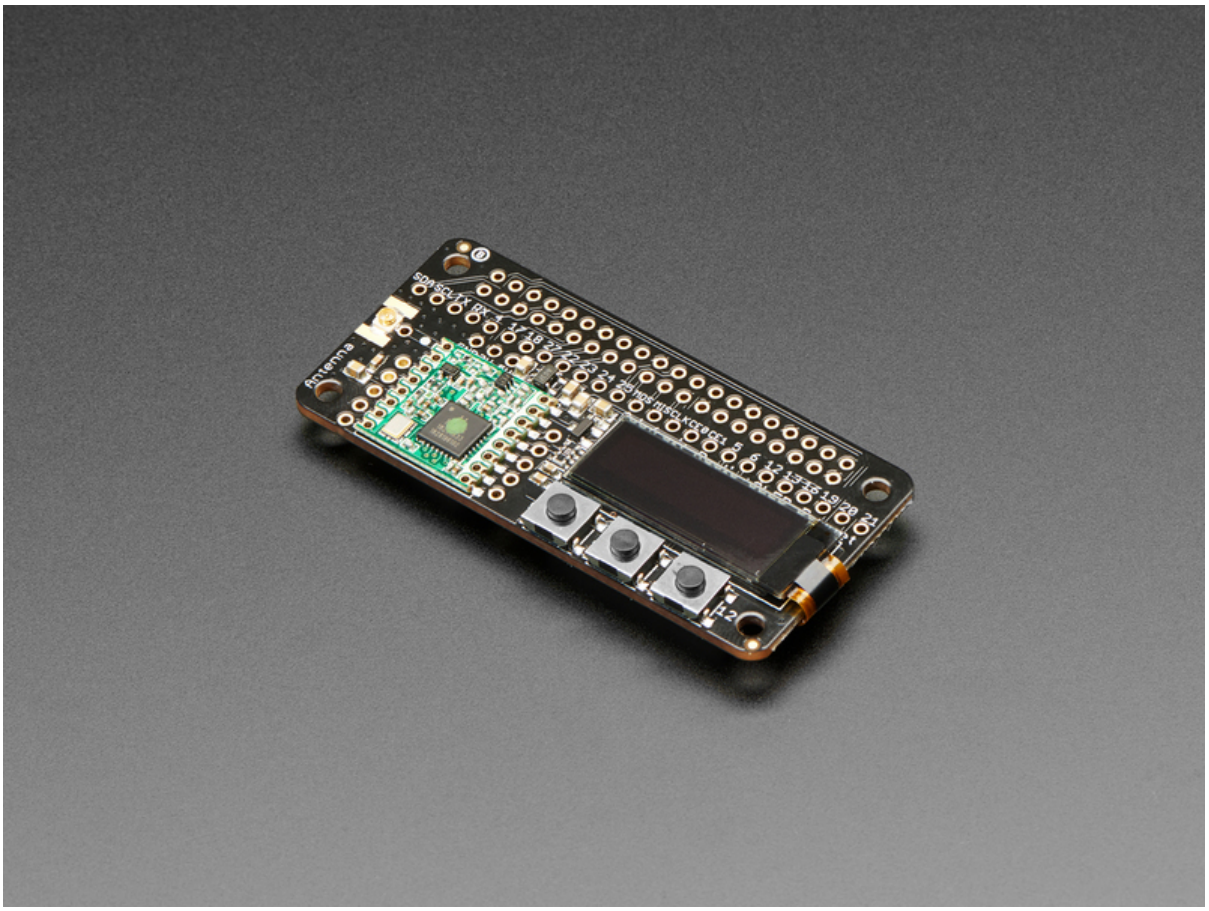
# Adafruit Radio Bonnets with OLED Display - RFM69 or RFM9X

Created by Kattni Rembor




https://learn.adafruit.com/adafruit-radio-bonnets

Last updated on 2024-03-08 03:16:09 PM EST

# Table of Contents


Overview 3

- Radio Modules & Frequency Variants
- RFM69 Specs
- RFM9x Specs

Pinouts 7

- Radio Module Pins
- 128x32 OLED
- Buttons

Antenna Options 9

- Wire Antenna
- uFL Antenna
- SMA Edge-Mount Antenna

RFM69 Raspberry Pi Setup 13

- Installing CircuitPython Libraries

Sending Data with the RFM69 16

- RFM69 and CircuitPython
- CircuitPython Transmitter/Receiver Example

LoRa Raspberry Pi Setup 19

- Installing CircuitPython Libraries

Sending Data with LoRa 22

- RFM9x and CircuitPython
- CircuitPython Transmitter/Receiver Example

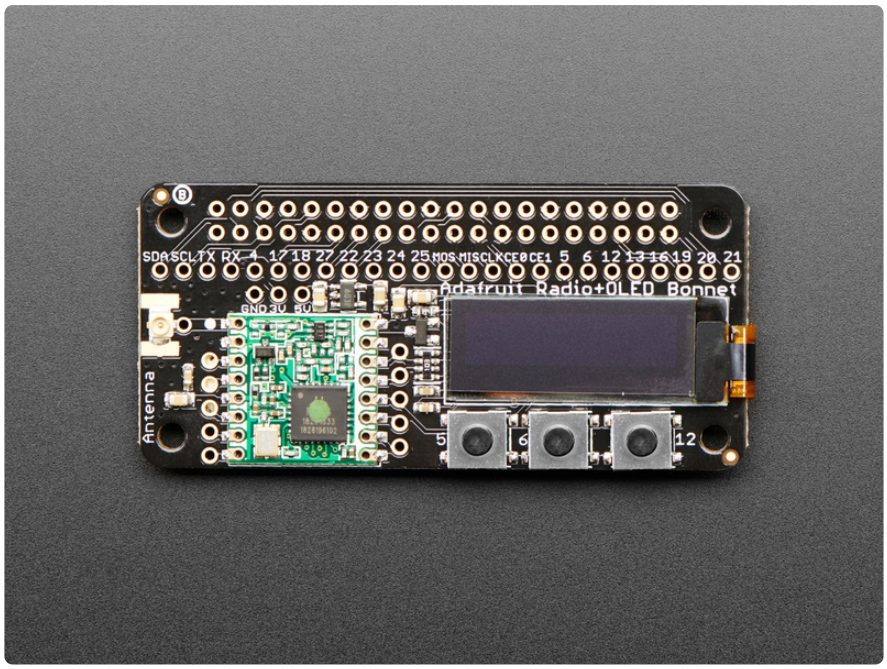
LoRaWAN Node Guide 26

LoRaWAN Pi Gateway Guide 26

Downloads 26

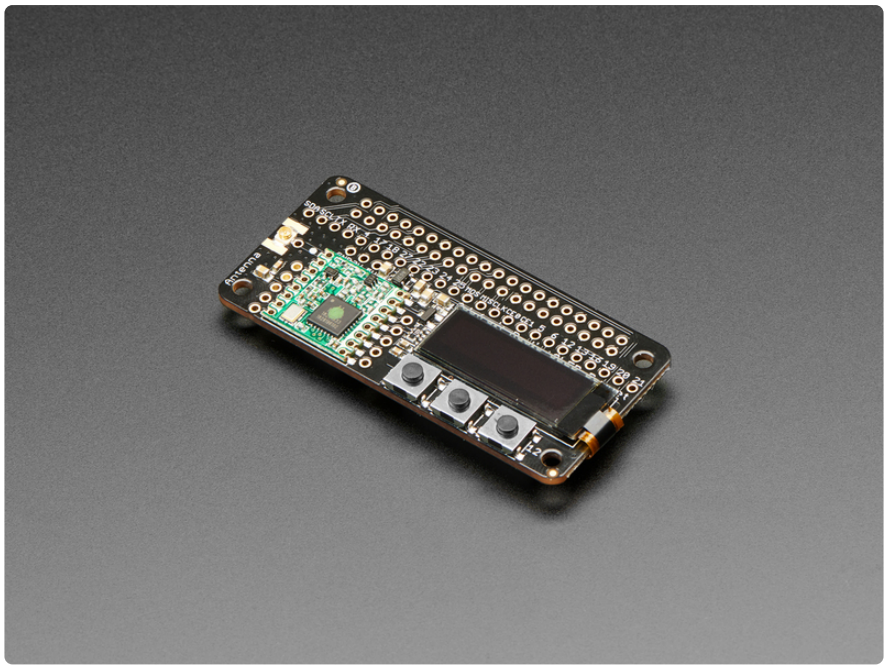
- Files
- Fab Print
- Schematic

# Overview



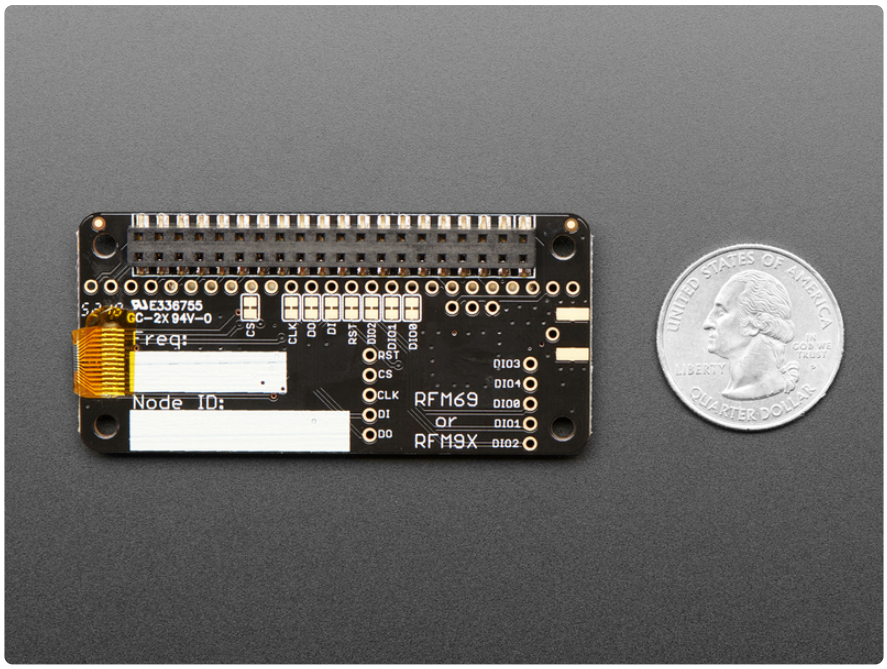

The latest Raspberry Pi computers come with WiFi and Bluetooth, and now you can add even more radio options with the Adafruit Radio Bonnets! Upgrade your Raspberry Pi with a radio so it can communicate over very long distances. These bonnets plug right into your Pi and give you long range wireless capabilities to remote nodes that may be battery powered. Or, you can create Internet gateways with ease.

You not only get a radio module, but also a 128x32 OLED display for status messages and three buttons you can use for creating a custom user interface or sending test messages. All of the above is supported with our Python libraries so you can send or receive radio data with other matching modules. With the LoRa Radio Bonnet, you send data to a LoRaWAN gateway, or even set up your own single channel LoRaWAN-to-Internet gateways.



Compared to the 2.4 GHz WiFi/Bluetooth radios on the Pi already, these modules run at 433 or 900 MHz (sub-GHz). You can't send data as fast but you can send data a lot farther. These packet radios are simpler than WiFi or BLE, you don't have to associate, pair, scan, or worry about connections. All you do is send data whenever you like, and any other modules tuned to that same frequency (and, with the same encryption key) will receive. The receiver can then send a reply back. The modules do packetization, error correction and can also auto-retransmit so its not like you have worry about everything but less power is wasted on maintaining a link or pairing.

These modules are great for use with other microcontrollers with matching radios (like say our RadioFruit Feathers (https://adafru.it/DFj)), say if you want a sensor node network or transmit data over a campus or town. The trade off is you need two or more radios, with matching frequencies.




## Radio Modules & Frequency Variants

These radio modules come in four variants (two modulation types and two frequencies):

- The first variant is the **RFM69 Radio** Bonnet. RFM69's are easiest to work with, and are well known and understood. It is available in **433MHz** or **900MHz** frequency ranges
- The second variant is the **LoRa Radio** Bonnet - an exciting and more powerful radio module, but also more expensive. It is available in **433MHz** or **900MHz** frequency ranges

Here are the four bonnets you can choose from. All radios are sold individually and can only talk to radios of the same part number. E.g. RFM69 900 MHz can only talk to RFM69 900 MHz, LoRa 433 MHz can only talk to LoRa 433, etc.

1. **RFM69 @ 433 MHz** - basic packetized FSK/GFSK/MSK/GMSK/OOK radio at 433 MHz for use in Europe ITU 1 license-free ISM, or for amateur use with restrictions (check your local  amateur regulations!)
2. **RFM69 @ 900 MHz** - basic packetized FSK/GFSK/MSK/GMSK/OOK radio at 868 or 915 MHz for use in Americas ITU 2 license-free ISM, or for amateur use with restrictions (check your amateur regulations!)
3. **RFM9x @ 433 MHz** - LoRa capable radio at 433 MHz for use in Europe ITU 1 license-free ISM, or for amateur use with restrictions (check your local amateur regulations!)

4. **RFM9x @ 900 MHz** - LoRa capable radio at 868 or 915 MHz for use in Americas ITU 2 license-free ISM, or for amateur use with restrictions (check your local amateur regulations!)

**Please note!** The 900 MHz radio version, can be used for either 868MHz or 915MHz transmission/reception. The exact radio frequency is determined when you load the software since it can be tuned around dynamically.

The radio modules themselves have the same pinout so the PCB is the same, but the library usage and wiring may vary. All use SPI for interfacing, and there are CircuitPython libraries available for both.

## RFM69 Specs

- SX1231 based module with SPI interface
- Packet radio with ready-to-go Arduino libraries
- Uses the license-free ISM bands
- +13 to +20 dBm up to 100 mW Power Output Capability (power output selectable in software)
- 50mA (+13 dBm) to 150mA (+20dBm) current draw for transmissions
- Range of approx. 350 meters, depending on obstructions, frequency, antenna and power output
- Create multipoint networks with individual node addresses
- Encrypted packet engine with AES-128

## RFM9x Specs

- SX127x LoRa® based module with SPI interface
- Packet radio with ready-to-go Arduino libraries
- Uses the license-free ISM bands
- +5 to +20 dBm up to 100 mW Power Output Capability (power output selectable in software)
- ~300uA during full sleep, ~120mA peak during +20dBm transmit, ~40mA during active radio listening.
- Our initial tests with default library settings: over 1.2mi/2Km line-of-sight with wire quarter-wave antennas. (With setting tweaking and directional antennas, 20Km is possible (https://adafru.it/mGa)).

Each bonnet comes fully assembled and ready to go. You can attach an antenna via the uFL connector, or cut and solder on a small piece of wire (any solid or stranded core is fine) in order to create your antenna.

# Pinouts

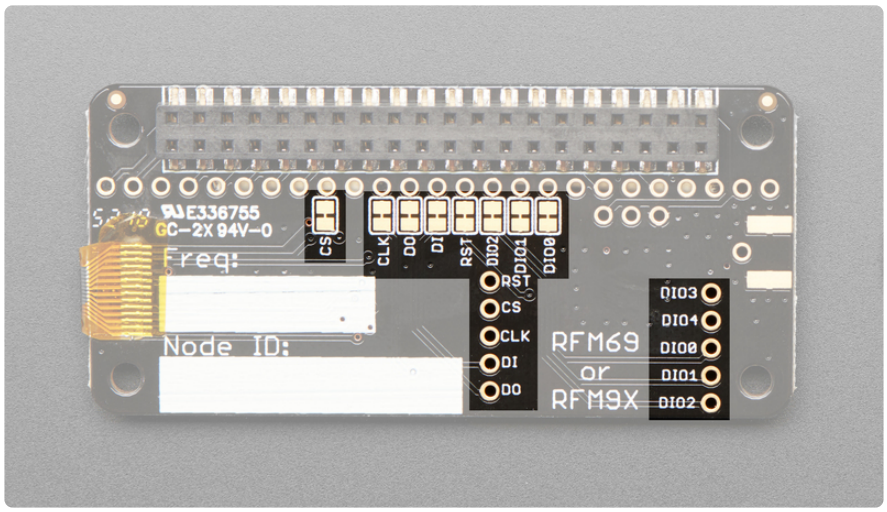
There are 4 different Radio Bonnets, with RFM69 or RFM9X and with 900MHz or 433MHz - however the overall pinouts are identical for all four!

# Radio Module Pins

There's a radio module on each bonnet, which we connect to the Raspberry Pi SPI port plus some extra pins for controlling.

**We connect the radio module to a set of default pins that match our code and examples, you can cut the solder traces on the bottom and re-wire them but the code/examples will need to be updated as well.**

Each radio has the same pin names and connections. Its easiest to see the pin numbering on the bottom.

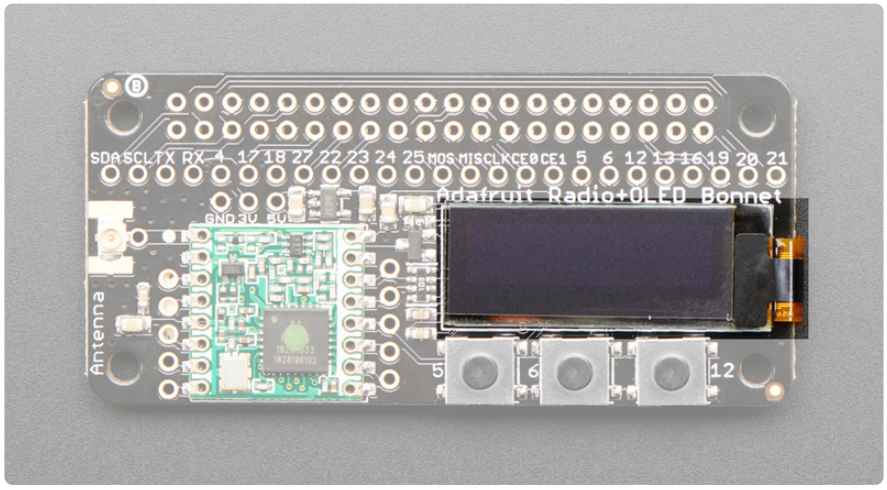



The two sets of pins here are broken out from the radio module. The defaults are:

- **RST** - Radio reset pin, connected to **GPIO25** on the Pi. It's pulled high by default which is reset. Pull LOW to turn on the radio.
- **CS** - Radio SPI Chip Select pin, connected to SPI **CE1** on the Pi
- **CLK** - Radio SPI Clock pin, connected to SPI **SCLK** on the Pi
- **DI** - Radio SPI data in pin, connected to SPI **MOSI** on the Pi
- **DO** - Radio SPI data out pin, connected to SPI **MISO** on the Pi
- **DIO0** - Radio digital IO #0 pin, we use this for status or IRQs. It's required for all our examples. Connected to **GPIO 22** on the Pi

- **DIO1** - Radio digital IO #1 pin, we use this for status. This is not used for our basic CircuitPython code, but is used by some more advanced libraries. You can cut this trace if you want to use the Pi pin for other devices. Connected to **GPIO 23** on the Pi
- **DIO2** - Radio digital IO #2 pin, we use this for status. This is not used for our basic CircuitPython code, but is used by some more advanced libraries. You can cut this trace if you want to use the Pi pin for other devices. Connected to **GPIO 24** on the Pi
- **DIO3** - Radio digital IO #3, not connected or used at this time.
- **DIO4** - Radio digital IO #3, not connected or used at this time.

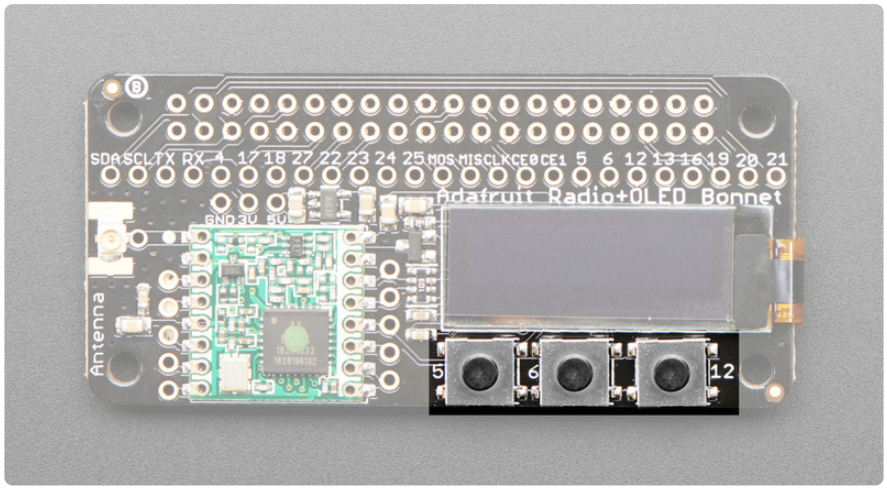
# 128x32 OLED




This bonnet comes with a 128x32 pixel OLED! **The OLED is connected via I2C**.

- **SCL** is connected to SCL on the Pi.
- **SDA** is connected to SDA on the Pi.

# Buttons




This bonnet comes with 3 buttons below the OLED. In order from left to right:

- **Button 1**: Connected to **GPIO 5** on the Pi
- **Button 2**: Connected to **GPIO 6** on the Pi
- **Button 3**: Connected to **GPIO 12** on the PI

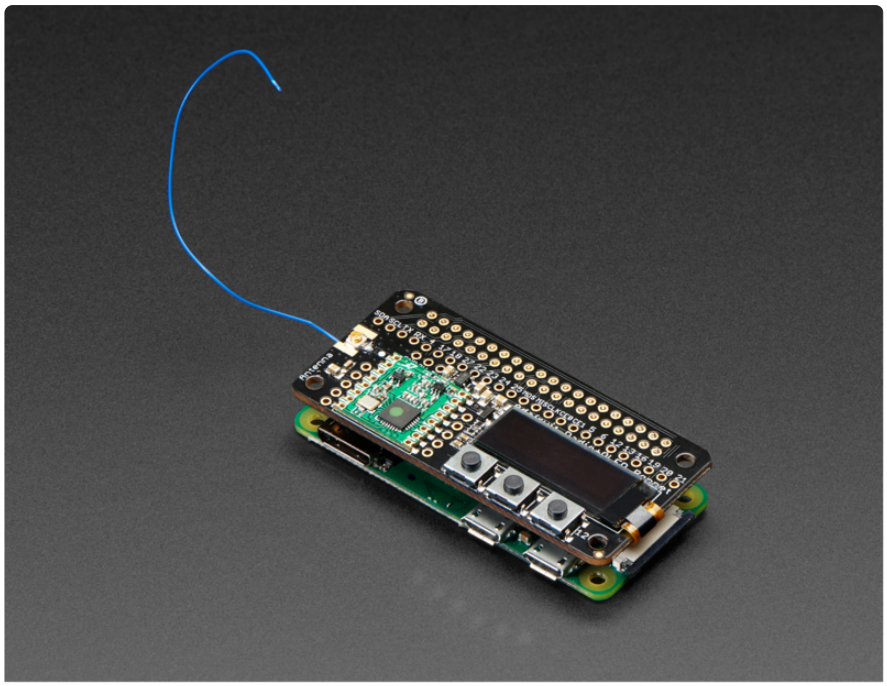
---

# Antenna Options

The Radio Bonnets do not have a built-in antenna. Instead, they have three options for attaching an antenna - for most low cost radio nodes, a wire works great. If you need to put the radio into an enclosure. We also include a uFL connector, pre-soldered, so you can use a uFL to SMA adapter to let you attach an external antenna. You can also solder an SMA edge-mount connector directly

## Wire Antenna

A wire antenna, aka "quarter wave whip antenna" is low cost and works very well! You just have to cut the wire down to the right length.



Cut a stranded or solid core wire the the proper length for the module/frequency

- **433 MHz** - 6.5 inches, or 16.5 cm
- **868 MHz** - 3.25 inches or 8.2 cm
- **915 MHz** - 3 inches or 7.8 cm

Strip a mm or two off the end of the wire, tin and solder into the **ANT** pad.

## uFL Antenna

If you want an external antenna that is a few inches away from the radio, you'll want a uFL antenna. The radio bonnets have a uFL connector soldered on already - you'll also need a uFL to SMA adapter (http://adafru.it/851) (or whatever adapter you need for the antenna you'll be using, SMA is the most common

Of course, you will also need an antenna of some sort, that matches your radio frequency. We have an antenna kit available which works well with the Radio Bonnet:

900Mhz Antenna Kit - For LoPy, LoRa, etc
This LoRa Antenna Kit is meant for use with the LoPy LoRa, WiFi and BLE board or the SiPy Sigfox,...
https://www.adafruit.com/product/3340

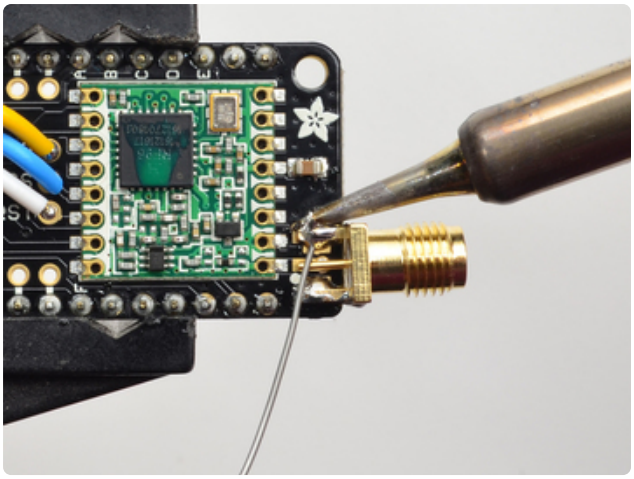
uFL connectors are rated for 30 connection cycles, but be careful when connecting/disconnecting to not rip the pads off the PCB. Once a uFL/SMA adapter is connected, use strain relief!

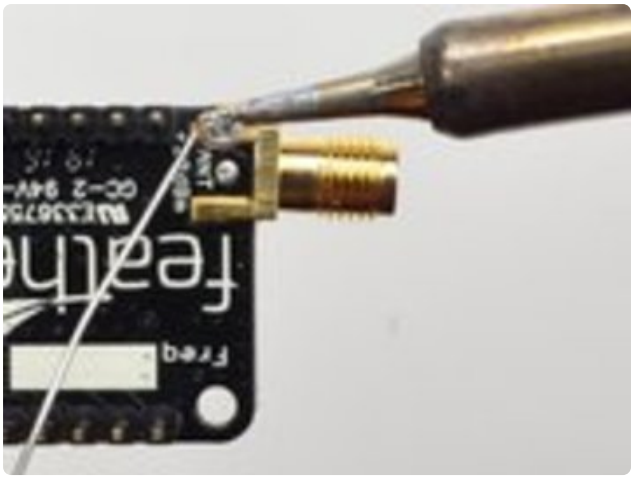
## SMA Edge-Mount Antenna

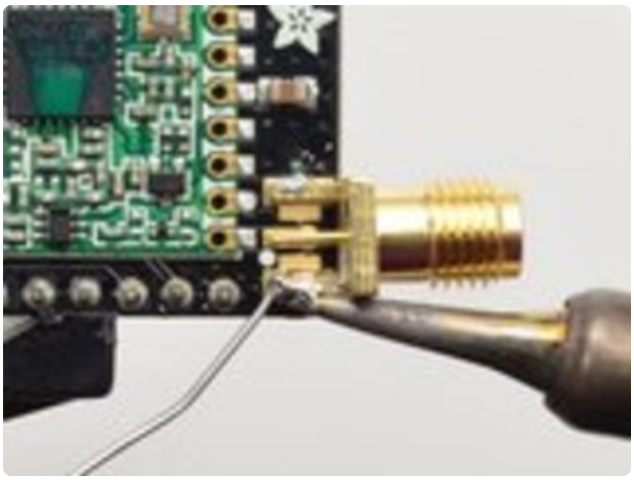
These strong edge connectors are used for many 'duck' antennas, and can also be panel mounted

You'll need an SMA (or, if you need RP-SMA for some reason) Edge-Mount connector with 1.6mm spacing

The SMA connector 'slides on' the top of the PCB

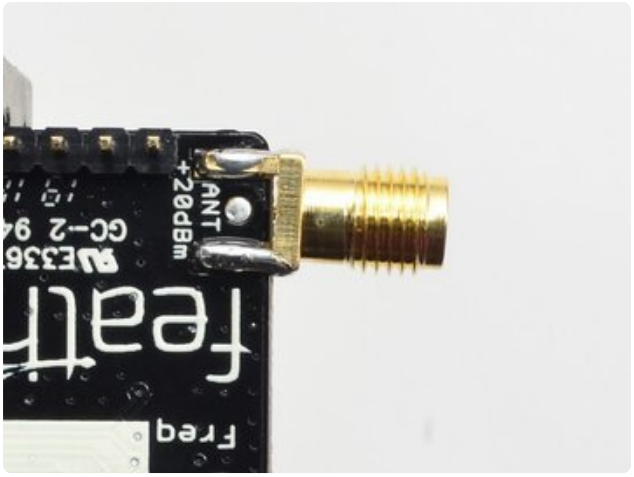






You'll need an SMA (or, if you need RP-SMA for some reason) Edge-Mount connector with 1.6mm spacing

The SMA connector 'slides on' the top of the PCB




Use plenty of solder to make sure you have a good strong mechanical connection. The duck antennas are long and make great levers, so they could pry apart the solder joints if not soldered well

---

# RFM69 Raspberry Pi Setup

This guide assumes that you've gotten your Raspberry Pi up and running, and have CircuitPython installed.

- If you have not done this yet, visit the installation guide **here** and come back when you're set up. (https://adafru.it/Deo)

## Installing CircuitPython Libraries

We're running CircuitPython on the Raspberry Pi, installing the libraries for radio communication is simple.

To **install the library for the display**, **enter the following into the terminal:**

```
pip3 install adafruit-circuitpython-ssd1306
```

You'll also need to **install the framebuf module** in order to write to the display.

```
sudo pip3 install adafruit-circuitpython-framebuf
```

To install the library for the **RFM69HCW** Module, **enter the following into the terminal:**

```
sudo pip3 install adafruit-circuitpython-rfm69
```

## RFM69 Connection Test!

Do not use this if you have a RFM9x Radio

The following code is for checking if the RFM69 radio is set up for transmitting and receiving. **Save the code on your Pi** as `rfm69_check.py`.

```
# SPDX-FileCopyrightText: 2018 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Wiring Check, Pi Radio w/RFM69

Learn Guide: https://learn.adafruit.com/lora-and-lorawan-for-raspberry-pi
Author: Brent Rubell for Adafruit Industries
"""
import time
import busio
from digitalio import DigitalInOut, Direction, Pull
import board
# Import the SSD1306 module.
import adafruit_ssd1306
# Import the RFM69 radio module.
import adafruit_rfm69

# Button A
btnA = DigitalInOut(board.D5)
btnA.direction = Direction.INPUT
btnA.pull = Pull.UP

# Button B
btnB = DigitalInOut(board.D6)
btnB.direction = Direction.INPUT
btnB.pull = Pull.UP

# Button C
btnC = DigitalInOut(board.D12)
btnC.direction = Direction.INPUT
btnC.pull = Pull.UP

# Create the I2C interface.
i2c = busio.I2C(board.SCL, board.SDA)

# 128x32 OLED Display
reset_pin = DigitalInOut(board.D4)
display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)
# Clear the display.
display.fill(0)
display.show()
width = display.width
height = display.height


# RFM69 Configuration
CS = DigitalInOut(board.CE1)
RESET = DigitalInOut(board.D25)
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

while True:
    # Draw a black filled box to clear the image.
    display.fill(0)
```

```
    # Attempt to set up the RFM69 Module
    try:
        rfm69 = adafruit_rfm69.RFM69(spi, CS, RESET, 915.0)
        display.text('RFM69: Detected', 0, 0, 1)
    except RuntimeError as error:
        # Thrown on version mismatch
        display.text('RFM69: ERROR', 0, 0, 1)
        print('RFM69 Error: ', error)

    # Check buttons
    if not btnA.value:
        # Button A Pressed
        display.text('Ada', width-85, height-7, 1)
        display.show()
        time.sleep(0.1)
    if not btnB.value:
        # Button B Pressed
        display.text('Fruit', width-75, height-7, 1)
        display.show()
        time.sleep(0.1)
    if not btnC.value:
        # Button C Pressed
        display.text('Radio', width-65, height-7, 1)
        display.show()
        time.sleep(0.1)

    display.show()
    time.sleep(0.1)
```

To use the code, enter the following in your terminal:

```
python3 rfm69_check.py
```

You'll also want to download the font file, `font5x8.bin`, and copy it into the same directory as the script:

**font5x8.bin**

https://adafru.it/DvA

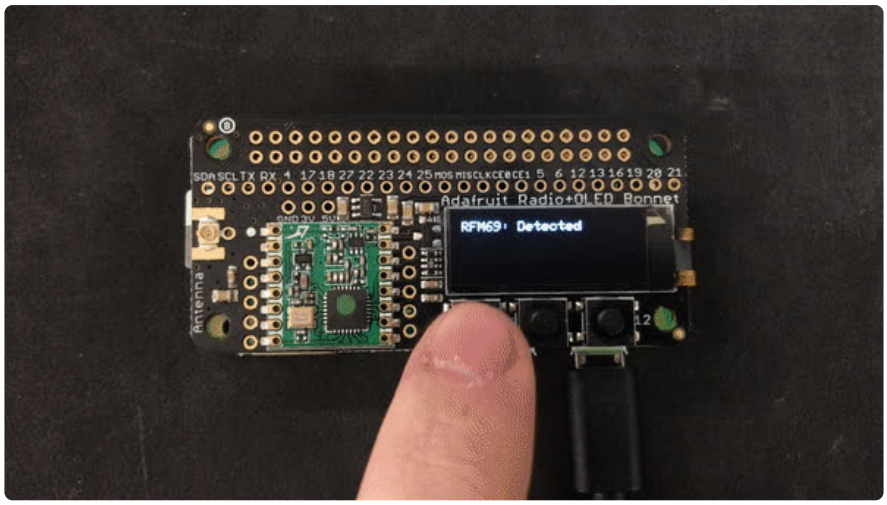
If you want to download this file to your Raspberry Pi via the command line, enter the following `wget` command into your terminal:

```
wget -O font5x8.bin https://github.com/adafruit/Adafruit_CircuitPython_framebuf/
blob/main/examples/font5x8.bin?raw=true
```

Now to check the setup:

The radio should display that a RFM69 module is detected. Pressing each of the buttons should display different text on the screen.

If it can't detect a RFM69 module, the screen will display `RFM69: ERROR`




With everything working, let's move on to using the radio.

---

# Sending Data with the RFM69

Do not use this if you have a RFM9x LoRa Radio.

To demonstrate how the RFM69 module sends packets, we'll build an example where one radio transmits (the Pi) and the other radio connected to a Feather receives the incoming transmission.

## RFM69 and CircuitPython

It's easy to use the RFM69HCW radio with CircuitPython and the Adafruit CircuitPython RFM69 (https://adafru.it/BjE) library.  This library allows you to easily write Python code that sends and receives packets of data with the radio.

Be careful to note **this code is for the RFM69 radio only** and **will not work with the RFM9X LoRa radios!**

You'll also need another radio with a RFM69HCW Module to run the example below. Make sure you have two of the **same type of radio** (i.e: RFM69HCW) **and the same frequency** (i.e: 915MHz).

## CircuitPython Transmitter/Receiver Example

Below is an example of using the RFM69HCW to transmit, or receive from, another RFM69HCW radio. Save this as **radio_rfm69.py** on your Raspberry Pi.

```
# SPDX-FileCopyrightText: 2018 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Example for using the RFM69HCW Radio with Raspberry Pi.

Learn Guide: https://learn.adafruit.com/lora-and-lorawan-for-raspberry-pi
Author: Brent Rubell for Adafruit Industries
"""
# Import Python System Libraries
import time
# Import Blinka Libraries
import busio
from digitalio import DigitalInOut, Direction, Pull
import board
# Import the SSD1306 module.
import adafruit_ssd1306
# Import the RFM69 radio module.
import adafruit_rfm69

# Button A
btnA = DigitalInOut(board.D5)
btnA.direction = Direction.INPUT
btnA.pull = Pull.UP

# Button B
btnB = DigitalInOut(board.D6)
btnB.direction = Direction.INPUT
btnB.pull = Pull.UP

# Button C
btnC = DigitalInOut(board.D12)
btnC.direction = Direction.INPUT
btnC.pull = Pull.UP

# Create the I2C interface.
i2c = busio.I2C(board.SCL, board.SDA)

# 128x32 OLED Display
reset_pin = DigitalInOut(board.D4)
display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)
# Clear the display.
display.fill(0)
display.show()
width = display.width
height = display.height

# Configure Packet Radio
CS = DigitalInOut(board.CE1)
RESET = DigitalInOut(board.D25)
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
rfm69 = adafruit_rfm69.RFM69(spi, CS, RESET, 915.0)
prev_packet = None
# Optionally set an encryption key (16 byte AES key). MUST match both
# on the transmitter and receiver (or be set to None to disable/the default).
rfm69.encryption_key =
b'\x01\x02\x03\x04\x05\x06\x07\x08\x01\x02\x03\x04\x05\x06\x07\x08'

while True:
    packet = None
    # draw a box to clear the image
    display.fill(0)
    display.text('RasPi Radio', 35, 0, 1)

    # check for packet rx
    packet = rfm69.receive()
    if packet is None:
```

```
        display.show()
        display.text('- Waiting for PKT -', 15, 20, 1)
    else:
        # Display the packet text and rssi
        display.fill(0)
        prev_packet = packet
        packet_text = str(prev_packet, "utf-8")
        display.text('RX: ', 0, 0, 1)
        display.text(packet_text, 25, 0, 1)
        time.sleep(1)

    if not btnA.value:
        # Send Button A
        display.fill(0)
        button_a_data = bytes("Button A!\r\n","utf-8")
        rfm69.send(button_a_data)
        display.text('Sent Button A!', 25, 15, 1)
    elif not btnB.value:
        # Send Button B
        display.fill(0)
        button_b_data = bytes("Button B!\r\n","utf-8")
        rfm69.send(button_b_data)
        display.text('Sent Button B!', 25, 15, 1)
    elif not btnC.value:
        # Send Button C
        display.fill(0)
        button_c_data = bytes("Button C!\r\n","utf-8")
        rfm69.send(button_c_data)
        display.text('Sent Button C!', 25, 15, 1)

    display.show()
    time.sleep(0.1)
```

To run the example, **enter the following into the terminal**:

```
python3 radio_rfm69.py
```

You'll also want to download the font file, `font5x8.bin`, and copy it into the same directory as the script:

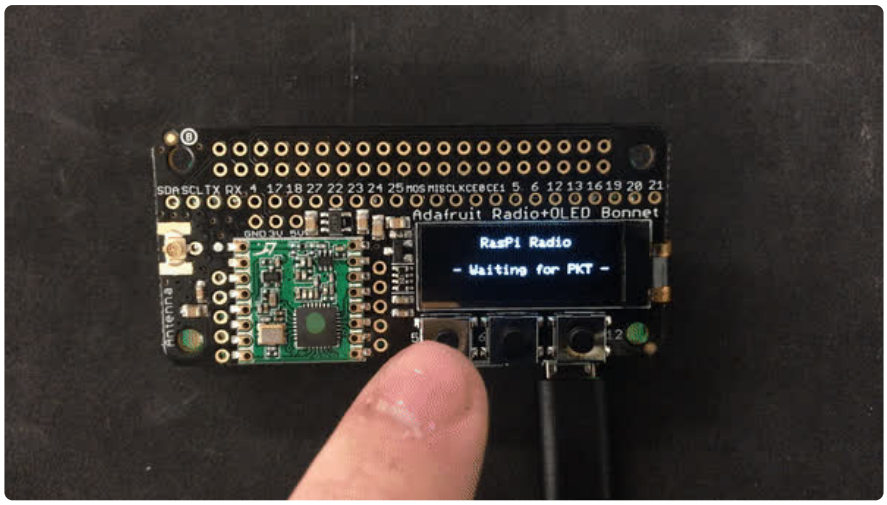
**font5x8.bin**

https://adafru.it/DvA

Both of the radios will listen for a new incoming packet. When they receive a new packet, they'll print the text from the packet to the display and to the terminal.

Press any of the three buttons to send data between radios. Pressing button a will send a packet with data 'Button A!' to the other radio, and so on!

To send a packet using the Pi, **press Button A**. You should see the text change to Sent Packet!




# LoRa Raspberry Pi Setup

This guide assumes that you've gotten your Raspberry Pi up and running, and have CircuitPython installed.

- If you have not done this yet, visit the installation guide **here** and come back when you're set up. (https://adafru.it/Deo)

## Installing CircuitPython Libraries

We're running CircuitPython on the Raspberry Pi, installing the libraries for radio communication is simple.

To **install the library for the display**, **enter the following into the terminal:**

```
sudo pip3 install adafruit-circuitpython-ssd1306
```

You'll also need to **install the framebuf module** in order to write to the display.

```
sudo pip3 install adafruit-circuitpython-framebuf
```

To install the library for the **RFM9x** Module, **enter the following into the terminal:**

```
sudo pip3 install adafruit-circuitpython-rfm9x
```

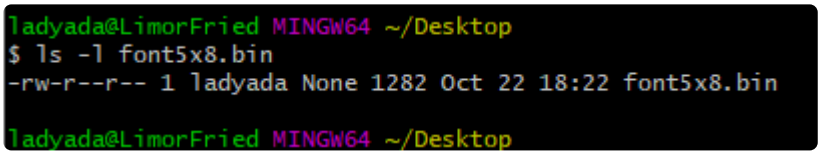
You'll also want to download the font file, `font5x8.bin`, and copy it into the same directory as the script

`wget https://github.com/adafruit/Adafruit_CircuitPython_framebuf/raw/main/examples/font5x8.bin`

**Click to download font5x8.bin**

https://adafru.it/DvA

Make sure the font file is 1282 bytes long, if not something went wrong with the download




## RFM9x Connection Test!

Do not use this if you have a RFM69 Radio

The following code is for checking if the RFM9x radio is set up for transmitting and receiving. **Save the code on your Pi** (save this to a location you can remember, like your Desktop or Downloads folder) as **`rfm9x_check.py`**.

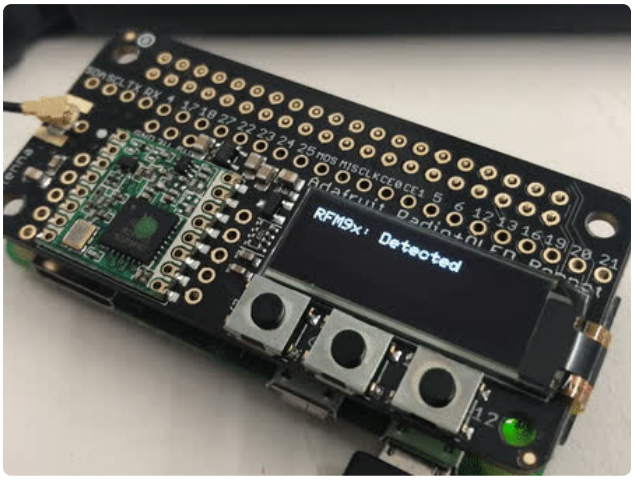
```
# SPDX-FileCopyrightText: 2018 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Wiring Check, Pi Radio w/RFM9x

Learn Guide: https://learn.adafruit.com/lora-and-lorawan-for-raspberry-pi
Author: Brent Rubell for Adafruit Industries
"""
import time
import busio
from digitalio import DigitalInOut, Direction, Pull
import board
# Import the SSD1306 module.
import adafruit_ssd1306
# Import the RFM9x radio module.
import adafruit_rfm9x

# Button A
btnA = DigitalInOut(board.D5)
btnA.direction = Direction.INPUT
btnA.pull = Pull.UP

# Button B
btnB = DigitalInOut(board.D6)
btnB.direction = Direction.INPUT
btnB.pull = Pull.UP

# Button C
btnC = DigitalInOut(board.D12)
btnC.direction = Direction.INPUT
btnC.pull = Pull.UP
```

```
# Create the I2C interface.
i2c = busio.I2C(board.SCL, board.SDA)

# 128x32 OLED Display
reset_pin = DigitalInOut(board.D4)
display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)
# Clear the display.
display.fill(0)
display.show()
width = display.width
height = display.height

# Configure RFM9x LoRa Radio
CS = DigitalInOut(board.CE1)
RESET = DigitalInOut(board.D25)
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

while True:
    # Clear the image
    display.fill(0)

    # Attempt to set up the RFM9x Module
    try:
        rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, 915.0)
        display.text('RFM9x: Detected', 0, 0, 1)
    except RuntimeError as error:
        # Thrown on version mismatch
        display.text('RFM9x: ERROR', 0, 0, 1)
        print('RFM9x Error: ', error)

    # Check buttons
    if not btnA.value:
        # Button A Pressed
        display.text('Ada', width-85, height-7, 1)
        display.show()
        time.sleep(0.1)
    if not btnB.value:
        # Button B Pressed
        display.text('Fruit', width-75, height-7, 1)
        display.show()
        time.sleep(0.1)
    if not btnC.value:
        # Button C Pressed
        display.text('Radio', width-65, height-7, 1)
        display.show()
        time.sleep(0.1)

    display.show()
    time.sleep(0.1)
```

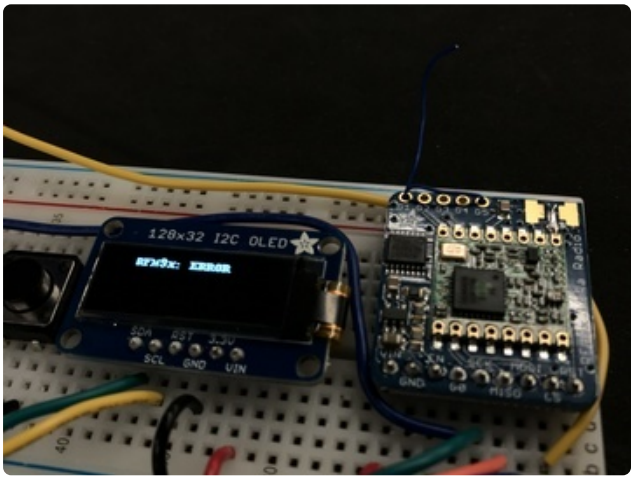
To use the code, copy and paste the following command into your terminal:

```
python3 rfm9x_check.py
```

Now to check the setup:




If the RFM9x/RFM69 is **detected**, the OLED will display **Detected**. You can test the buttons by pressing them.




**If the wiring of the radio module is incorrect** - the display will show **ERROR**. Check over your wiring on the Wiring Page and re-run the test. You may also need to ensure the correct CircuitPython library is installed for the module.

**If the OLED does not turn on** - first check that it is wired correctly. Then, make sure you enabled I2C from `raspi-config` and installed the required libraries (`adafruit-circuitpython-framebuf` and `adafruit-circuitpython-ssd1306`).

With everything working, let's move on to using the radio.

---

# Sending Data with LoRa

Do not use this if you have a RFM69 Packet Radio, its for RFM9X radio's only!

The RFM9x is a more expensive module than the RFM69, but it has a trick - LoRa. LoRa is a spread spectrum modulation technique patented by Semtech (https://adafru.it/DuM). It allows your packets to be sent over farther distances (a few km in a city like New York and around 40km in a rural area).

It's also a low power protocol with batteries lasting in the year range, instead of a few days with WiFi. This is made possible by only powering up the LoRa radio when packets are being sent, instead of keeping the radio always-on (like the WiFi radio on your cell phone).

When building a project which uses LoRa, keep in mind that only a few hundred bytes per-transmission can be sent, and that each transmission will cause the battery life to decrease.

If a LoRa project involves a lot of sensors, expect to cram a lot of data into a small packet which is transmitted infrequently.

## RFM9x and CircuitPython

It's easy to use the RFM9x LoRa radio with CircuitPython and the Adafruit CircuitPython RFM9x (https://adafru.it/BjD) module.  This module allows you to easily write Python code that sends and receives packets of data with the radio.

Be careful to note **this library is for the RFM9x (RFM95/96/97/98) Radio only** and **will not work** with the RFM69.

To demonstrate how the RFM9x LoRa module sends packets, we'll build an example where we send and receive data between two radios.

## CircuitPython Transmitter/Receiver Example

Below is an example of using the RFM9x to transmit, or receive from, another RFM9x radio. Save this as **radio_rfm9x.py** on your Raspberry Pi.

```
# SPDX-FileCopyrightText: 2018 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Example for using the RFM9x Radio with Raspberry Pi.

Learn Guide: https://learn.adafruit.com/lora-and-lorawan-for-raspberry-pi
Author: Brent Rubell for Adafruit Industries
"""
# Import Python System Libraries
import time
# Import Blinka Libraries
import busio
from digitalio import DigitalInOut, Direction, Pull
import board
# Import the SSD1306 module.
import adafruit_ssd1306
# Import RFM9x
import adafruit_rfm9x

# Button A
btnA = DigitalInOut(board.D5)
btnA.direction = Direction.INPUT
btnA.pull = Pull.UP

# Button B
btnB = DigitalInOut(board.D6)
btnB.direction = Direction.INPUT
btnB.pull = Pull.UP

# Button C
btnC = DigitalInOut(board.D12)
btnC.direction = Direction.INPUT
btnC.pull = Pull.UP

# Create the I2C interface.
i2c = busio.I2C(board.SCL, board.SDA)

# 128x32 OLED Display
reset_pin = DigitalInOut(board.D4)
display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)
# Clear the display.
display.fill(0)
display.show()
width = display.width
height = display.height

# Configure LoRa Radio
CS = DigitalInOut(board.CE1)
RESET = DigitalInOut(board.D25)
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, 915.0)
rfm9x.tx_power = 23
prev_packet = None

while True:
    packet = None
    # draw a box to clear the image
```

```
    display.fill(0)
    display.text('RasPi LoRa', 35, 0, 1)

    # check for packet rx
    packet = rfm9x.receive()
    if packet is None:
        display.show()
        display.text('- Waiting for PKT -', 15, 20, 1)
    else:
        # Display the packet text and rssi
        display.fill(0)
        prev_packet = packet
        packet_text = str(prev_packet, "utf-8")
        display.text('RX: ', 0, 0, 1)
        display.text(packet_text, 25, 0, 1)
        time.sleep(1)

    if not btnA.value:
        # Send Button A
        display.fill(0)
        button_a_data = bytes("Button A!\r\n","utf-8")
        rfm9x.send(button_a_data)
        display.text('Sent Button A!', 25, 15, 1)
    elif not btnB.value:
        # Send Button B
        display.fill(0)
        button_b_data = bytes("Button B!\r\n","utf-8")
        rfm9x.send(button_b_data)
        display.text('Sent Button B!', 25, 15, 1)
    elif not btnC.value:
        # Send Button C
        display.fill(0)
        button_c_data = bytes("Button C!\r\n","utf-8")
        rfm9x.send(button_c_data)
        display.text('Sent Button C!', 25, 15, 1)


    display.show()
    time.sleep(0.1)
```

You'll also want to download the font file, **`font5x8.bin`**, and copy it into the same directory as the script:

font5x8.bin

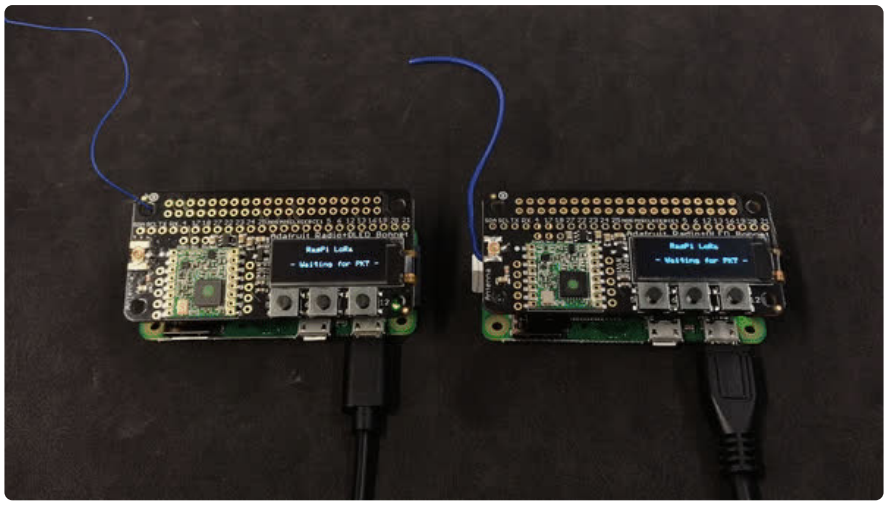
https://adafru.it/DvA

To run the example, **enter the following into the terminal**:

```
python3 radio_rfm9x.py
```

Both of the radios will listen for a new incoming packet. When they receive a new packet, they'll print the text from the packet to the display (and to the terminal).

Press any of the three buttons to send data between radios. Pressing button a will send a packet with data 'Button A!' to the other radio, and so on!



# LoRaWAN Node Guide

LoRaWAN Node Guide (https://adafru.it/DHq)

# LoRaWAN Pi Gateway Guide

LoRaWAN Pi Gateway Guide (https://adafru.it/DHr)

# Downloads

# Files

**RFM9x**

- SX127x Datasheet (http://adafru.it/31791276777879)- The RFM9X LoRa radio chip itself
- RFM9X (https://adafru.it/FTK) - The radio module, which contains the SX1272 chipset
- FCC Test Report (http://adafru.it/30789591522528132529)
- ETSI Test Report (http://adafru.it/30789586822528112529)
- RoHS Report (http://adafru.it/307814015141021513720O)

**RFM69**

- SX1231 Datasheet (http://adafru.it/30761231) - The RFM69 radio chip itself
- RFM69HCW datasheet (http://adafru.it/30766911)- contains the SX1231 datasheet plus details about the module (https://adafru.it/mFX)

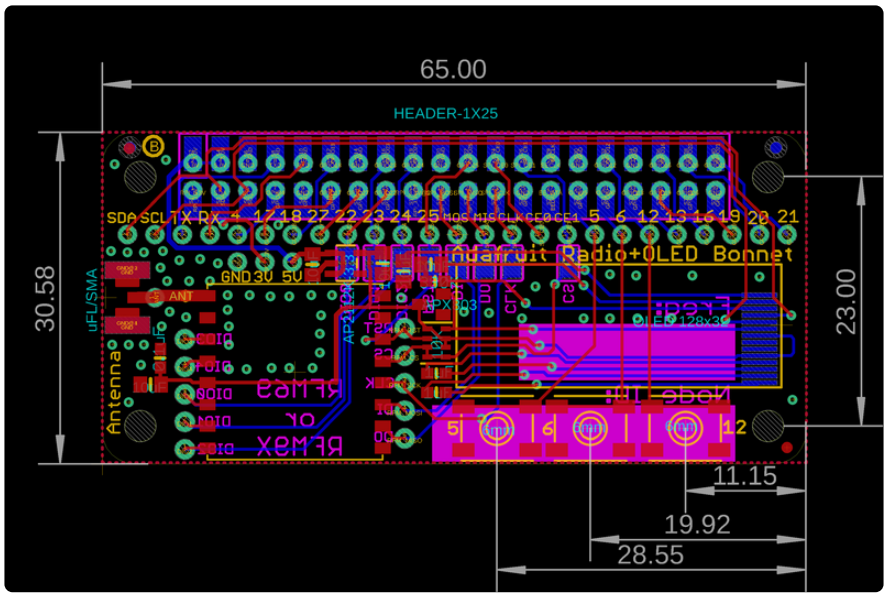
- RoHS Test Report (http://adafru.it/307614015141021513720 0)
- RoHS Test Report (http://adafru.it/30761503185404159830 75)
- REACH Test Report (http://adafru.it/30761503185407159830 75)
- ETSI Test Report (http://adafru.it/3076698682)
- FCC Test Report (http://adafru.it/307030703076699152)

**Radio Bonnet**

- Schematic and board files on GitHub (https://adafru.it/DFL) (Pinouts and layout are the same for all four radio versions)

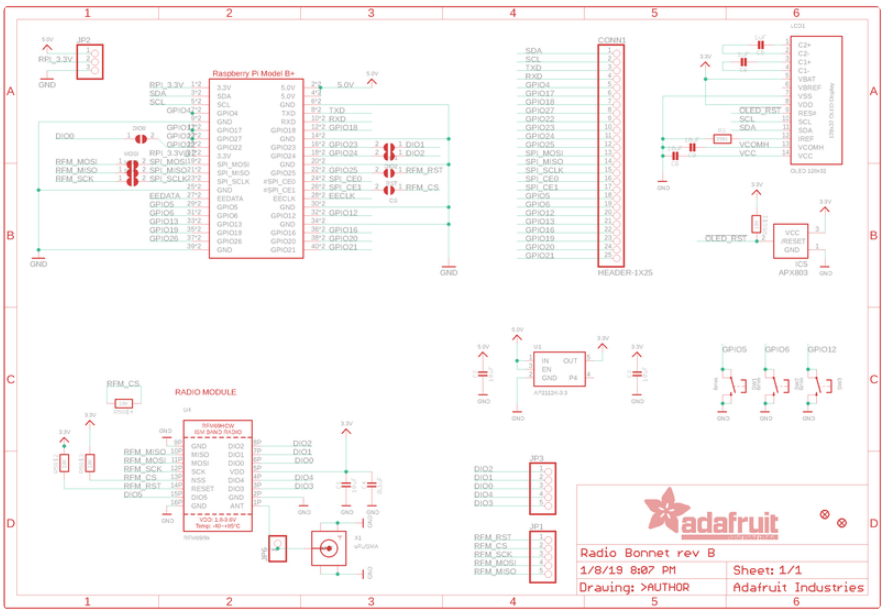
# Fab Print

(Pinouts and layout are the same for all four radio versions)




# Schematic

(Pinouts and layout are the same for all four radio versions)


Raspberry Pi Model B+
RADIO MODULE
HEADER-1X25
APX803
Radio Bonnet rev B
1/8/19 8:07 PM
Sheet: 1/1
Drawing: >AUTHOR
Adafruit Industries