AdafruitRFM69 Library Documentation

Release 1.0

Tony DiCola

Nov 19, 2018

Contents

CircuitPython RFM69 packet radio module. This supports basic RadioHead-compatible sending and receiving of packets with RFM69 series radios (433/915Mhz).

Note: This does NOT support advanced RadioHead features like guaranteed delivery–only 'raw' packets are currently supported.

Warning: This is NOT for LoRa radios!

Note: This is a 'best effort' at receiving data using pure Python code–there is not interrupt support so you might lose packets if they're sent too quickly for the board to process them. You will have the most luck using this in simple low bandwidth scenarios like sending and receiving a 60 byte packet at a time–don't try to receive many kilobytes of data at a time!

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython
- Bus Device

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER 2

Usage Example

See examples/rfm69_simpletest.py for a simple demo of the usage. Note: the default baudrate for the SPI is 5000000 (5MHz). The maximum setting is 10Mhz but transmission errors have been observed expecially when using breakout boards. For breakout boards or other configurations where the boards are separated, it may be necessary to reduce the baudrate for reliable data transmission. The baud rate may be specified as an keyword parameter when initializing the board. To set it to 1000000 use :

```
# Initialze RFM radio
rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, RADIO_FREQ_MHZ,baudrate=1000000)
```

CHAPTER 3

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 4

Building locally

To build this library locally you'll need to install the circuitpython-build-tools package.

```
python3 -m venv .env
source .env/bin/activate
pip install circuitpython-build-tools
```

Once installed, make sure you are in the virtual environment:

```
source .env/bin/activate
```

Then run the build:

```
circuitpython-build-bundles --filename_prefix adafruit-circuitpython-rfm69 --library_
↪location .
```

4.1 Sphinx documentation

Sphinx is used to build the documentation based on rST files and comments in the code. First, install dependencies (feel free to reuse the virtual environment from above):

```
python3 -m venv .env
source .env/bin/activate
pip install Sphinx sphinx-rtd-theme
```

Now, once you have the virtual environment activated:

```
cd docs
sphinx-build -E -W -b html . _build/html
```

This will output the documentation to `docs/_build/html`. Open the index.html in your browser to view them. It will also (due to -W) error out on any warning like Travis will. This is a good way to locally verify it will pass.

CHAPTER 5

Table of Contents

5.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/rfm69_simpletest.py

```
# Simple example to send a message and then wait indefinitely for messages
# to be received.  This uses the default RadioHead compatible GFSK_Rb250_Fd250
# modulation and packet format for the radio.
# Author: Tony DiCola
import board
import busio
import digitalio

import adafruit_rfm69


# Define radio parameters.
RADIO_FREQ_MHZ = 915.0  # Frequency of the radio in Mhz. Must match your
                        # module! Can be a value like 915.0, 433.0, etc.

# Define pins connected to the chip, use these if wiring up the breakout according to the guide:
CS = digitalio.DigitalInOut(board.D5)
RESET = digitalio.DigitalInOut(board.D6)
# Or uncomment and instead use these if using a Feather M0 RFM69 board
# and the appropriate CircuitPython build:
#CS = digitalio.DigitalInOut(board.RFM69_CS)
#RESET = digitalio.DigitalInOut(board.RFM69_RST)

# Initialize SPI bus.
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

```

(continues on next page)

(continued from previous page)

```
# Initialze RFM radio
rfm69 = adafruit_rfm69.RFM69(spi, CS, RESET, RADIO_FREQ_MHZ)

# 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'

# Print out some chip state:
print('Temperature: {0}C'.format(rfm69.temperature))
print('Frequency: {0}mhz'.format(rfm69.frequency_mhz))
print('Bit rate: {0}kbit/s'.format(rfm69.bitrate/1000))
print('Frequency deviation: {0}hz'.format(rfm69.frequency_deviation))

# Send a packet.  Note you can only send a packet up to 60 bytes in length.
# This is a limitation of the radio packet size, so if you need to send larger
# amounts of data you will need to break it into smaller send calls.  Each send
# call will wait for the previous one to finish before continuing.
rfm69.send(bytes('Hello world!\r\n',"utf-8"))
print('Sent hello world message!')

# Wait to receive packets.  Note that this library can't receive data at a fast
# rate, in fact it can only receive and process one 60 byte packet at a time.
# This means you should only use this for low bandwidth scenarios, like sending
# and receiving a single message at a time.
print('Waiting for packets...')
while True:
    packet = rfm69.receive()
    # Optionally change the receive timeout from its default of 0.5 seconds:
    #packet = rfm69.receive(timeout=5.0)
    # If no packet was received during the timeout then None is returned.
    if packet is None:
        print('Received nothing! Listening again...')
    else:
        # Received a packet!
        # Print out the raw bytes of the packet:
        print('Received (raw bytes): {0}'.format(packet))
        # And decode to ASCII text and print it too.  Note that you always
        # receive raw bytes and need to convert to a text format like ASCII
        # if you intend to do string processing on your data.  Make sure the
        # sending side is sending ASCII data before you try to decode!
        packet_text = str(packet, 'ascii')
        print('Received (ASCII): {0}'.format(packet_text))
```

5.2 adafruit_rfm69

CircuitPython RFM69 packet radio module. This supports basic RadioHead-compatible sending and receiving of packets with RFM69 series radios (433/915Mhz).

Note: This does NOT support advanced RadioHead features like guaranteed delivery–only 'raw' packets are currently supported.

Warning: This is NOT for LoRa radios!

Note: This is a 'best effort' at receiving data using pure Python code–there is not interrupt support so you might lose packets if they're sent too quickly for the board to process them. You will have the most luck using this in simple low bandwidth scenarios like sending and receiving a 60 byte packet at a time–don't try to receive many kilobytes of data at a time!

- Author(s): Tony DiCola

5.2.1 Implementation Notes

Hardware:

- Adafruit RFM69HCW Transceiver Radio Breakout - 868 or 915 MHz - RadioFruit (Product ID: 3070)
- Adafruit RFM69HCW Transceiver Radio Breakout - 433 MHz - RadioFruit (Product ID: 3071)
- Adafruit Feather M0 RFM69HCW Packet Radio - 868 or 915 MHz - RadioFruit (Product ID: 3176)
- Adafruit Feather M0 RFM69HCW Packet Radio - 433 MHz - RadioFruit (Product ID: 3177)
- Adafruit Radio FeatherWing - RFM69HCW 900MHz - RadioFruit (Product ID: 3229)
- Adafruit Radio FeatherWing - RFM69HCW 433MHz - RadioFruit (Product ID: 3230)

Software and Dependencies:

- Adafruit CircuitPython firmware for the ESP8622 and M0-based boards: https://github.com/adafruit/circuitpython/releases
- Adafruit's Bus Device library: https://github.com/adafruit/Adafruit_CircuitPython_BusDevice

class adafruit_rfm69.**RFM69**(*spi*, *cs*, *reset*, *frequency*, ***, *sync_word=b'-xd4'*, *preamble_length=4*, *encryption_key=None*, *high_power=True*, *baudrate=5000000*)

Interface to a RFM69 series packet radio. Allows simple sending and receiving of wireless data at supported frequencies of the radio (433/915mhz).

Parameters

- **spi** (*busio.SPI*) – The SPI bus connected to the chip. Ensure SCK, MOSI, and MISO are connected.
- **cs** (*DigitalInOut*) – A DigitalInOut object connected to the chip's CS/chip select line.
- **reset** (*DigitalInOut*) – A DigitalInOut object connected to the chip's RST/reset line.
- **frequency** (*int*) – The center frequency to configure for radio transmission and reception. Must be a frequency supported by your hardware (i.e. either 433 or 915mhz).
- **sync_word** (*bytes*) – A byte string up to 8 bytes long which represents the syncronization word used by received and transmitted packets. Read the datasheet for a full understanding of this value! However by default the library will set a value that matches the RadioHead Arduino library.
- **preamble_length** (*int*) – The number of bytes to pre-pend to a data packet as a preamble. This is by default 4 to match the RadioHead library.

- **encryption_key** (*bytes*) – A 16 byte long string that represents the AES encryption key to use when encrypting and decrypting packets. Both the transmitter and receiver MUST have the same key value! By default no encryption key is set or used.
- **high_power** (*bool*) – Indicate if the chip is a high power variant that supports boosted transmission power. The default is True as it supports the common RFM69HCW modules sold by Adafruit.

Note: The D0/interrupt line is currently unused by this module and can remain unconnected.

Remember this library makes a best effort at receiving packets with pure Python code. Trying to receive packets too quickly will result in lost data so limit yourself to simple scenarios of sending and receiving single packets at a time.

Also note this library tries to be compatible with raw RadioHead Arduino library communication. This means the library sets up the radio modulation to match RadioHead's default of GFSK encoding, 250kbit/s bitrate, and 250khz frequency deviation. To change this requires explicitly setting the radio's bitrate and encoding register bits. Read the datasheet and study the init function to see an example of this–advanced users only! Advanced RadioHead features like address/node specific packets or guaranteed delivery are not supported. Only simple broadcast of packets to all listening radios is supported. Features like addressing and guaranteed delivery need to be implemented at an application level.

bitrate
The modulation bitrate in bits/second (or chip rate if Manchester encoding is enabled). Can be a value from ~489 to 32mbit/s, but see the datasheet for the exact supported values.

encryption_key
The AES encryption key used to encrypt and decrypt packets by the chip. This can be set to None to disable encryption (the default), otherwise it must be a 16 byte long byte string which defines the key (both the transmitter and receiver must use the same key value).

frequency_deviation
The frequency deviation in Hertz.

frequency_mhz
The frequency of the radio in Megahertz. Only the allowed values for your radio must be specified (i.e. 433 vs. 915 mhz)!

idle()
Enter idle standby mode (switching off high power amplifiers if necessary).

listen()
Listen for packets to be received by the chip. Use *receive()* to listen, wait and retrieve packets as they're available.

operation_mode
The operation mode value. Unless you're manually controlling the chip you shouldn't change the operation_mode with this property as other side-effects are required for changing logical modes–use *idle()*, *sleep()*, *transmit()*, *listen()* instead to signal intent for explicit logical modes.

preamble_length
The length of the preamble for sent and received packets, an unsigned 16-bit value. Received packets must match this length or they are ignored! Set to 4 to match the RadioHead RFM69 library.

receive(*timeout=0.5*, *keep_listening=True*)
Wait to receive a packet from the receiver. Will wait for up to timeout_s amount of seconds for a packet to be received and decoded. If a packet is found the payload bytes are returned, otherwise None is returned (which indicates the timeout elapsed with no reception). Note this assumes a 4-byte header is prepended

to the data for compatibilty with the RadioHead library (the header is not validated nor returned). If keep_listening is True (the default) the chip will immediately enter listening mode after reception of a packet, otherwise it will fall back to idle mode and ignore any future reception.

reset()
Perform a reset of the chip.

rssi
The received strength indicator (in dBm) of the last received message.

send(*data*, *timeout=2.0*)
Send a string of data using the transmitter. You can only send 60 bytes at a time (limited by chip's FIFO size and appended headers). Note this appends a 4 byte header to be compatible with the RadioHead library. The timeout is just to prevent a hang (arbitrarily set to 2 seconds)

sleep()
Enter sleep mode.

sync_word
The synchronization word value. This is a byte string up to 8 bytes long (64 bits) which indicates the synchronization word for transmitted and received packets. Any received packet which does not include this sync word will be ignored. The default value is 0x2D, 0xD4 which matches the RadioHead RFM69 library. Setting a value of None will disable synchronization word matching entirely.

temperature
The internal temperature of the chip in degrees Celsius. Be warned this is not calibrated or very accurate.

> **Warning:** Reading this will STOP any receiving/sending that might be happening!

transmit()
Transmit a packet which is queued in the FIFO. This is a low level function for entering transmit mode and more. For generating and transmitting a packet of data use *send()* instead.

tx_power
The transmit power in dBm. Can be set to a value from -2 to 20 for high power devices (RFM69HCW, high_power=True) or -18 to 13 for low power devices. Only integer power levels are actually set (i.e. 12.5 will result in a value of 12 dBm).

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index